

AD-A055 902 SOFTECH INC WALTHAM MASS
OPERATIONAL SOFTWARE CONCEPT OSC EXECUTIVE EVALUATION/REFINEMENT--ETC(U)
AUG 77 M G WILLOUGHBY, C K HITCHON F33615-76-C-1192

UNCLASSIFIED

1025-3

AFAL-TR-77-87

NL

1 of 2
AD A055 902



AU NO.

DDC FILE COPY

ADA055902

(2)



A010415
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| | | | |
|--|-----------------------|---|--|
| (8) REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM | |
| REPORT NUMBER AFAL-TR-77-87 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER | |
| TITLE (and Subtitle) OPERATIONAL SOFTWARE CONCEPT EXECUTIVE EVALUATION/REFINEMENT OSC | | 9. TYPE OF REPORT & PERIOD COVERED Final Sep [redacted] - Dec [redacted] 76 PERFORMING ORG. REPORT NUMBER 1025-3 | |
| 10. CONTRACT OR GRANT NUMBER Michael G. Willoughby Carl K. Hitchon | | 15. F33615-76-C-1192 | |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS SoftTech, Inc. 460 Totten Pond Road Waltham, Massachusetts 02154 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6220UF 2003 05,14 1105 | |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory System Technology Branch (AAT) Wright-Patterson AFB, Ohio 45433 | | 12. REPORT DATE June 1977 | |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 11 Aug 77 | | 13. NUMBER OF PAGES 145 | |
| | | 15. SECURITY CLASS. (of this report) Unclassified | |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited. | | | |
| 17. DISTRIBUTION STATEMENT (of the contract entered in Block 10, if different from Report) 14 91025-3 12 JUN 78 P. DDC/GRADMLP JUN 30 1978 | | | |
| 18. SUPPLEMENTARY NOTES | | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Modular Software Avionics Software Directed Flow Graphs System Software Support Software Executive Software Higher Order Language Software Design | | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes executives built using the Operational Software Concept (OSC). These executives are designed to operate on a federated network of four DAIS processors connected by DAIS multiplex data busses. In fact, the executives and applications, represented by stubs, have been implemented on a two processor system. The applications supported by the executives are specified using Directed Flowgraphs (DFG) as described in Technical Report AFAL-TR-74-168, Volume II | | | |

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

DAD-A010 415. → (over)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

405 932

78 06 28 122

SLT

from OSC Phase I. It is assumed that the reader of this report will be familiar with the contents of the above-mentioned document.

The executives have been built based on the design presented in the OSC Computer Program Development Specification, Technical Report for April 1976 to September 1976. The baseline executives were coded on the DAIS laboratory PDP-10, primarily in J731, providing full generality for the DFG specified application. The DFG was provided by AFAL as representative of a DAIS mission. These executives had time overheads of 37.1% and 41.0% and space overheads of 36.5% and 33.7% for processors 0 (the head processor) and 1, respectively. The baseline executives were tuned primarily for High Order Language (HOL) inefficiencies, HOL deficiencies and generality reduction for the specific DFG. The resulting final tuned executives had time overheads of 11.1% and 11.3% and space overheads of 22.1% and 19.2% for processors 0 and 1, respectively. These overhead figures and the detailed statistics presented in the report represent the state of the executives on 20 December 1976 and are based on the DAIS processor described in Specification Number MN255R817-1 of September 1976 and the DAIS multiplex data bus described in Specification Number SA301300B-15 of March 1976.

This report is divided into three sections and an Appendix. The first section describes the process of building an executive based on a DFG. The second section describes the parameters affecting system performance that are associated with the DFG supported by the executives. The third section presents the baseline executive statistics, tuning method descriptions and statistics for the final tuned executive. The Appendix provides program listings of intermediate tuning results of the final tuned executives.

FOREWARD

This final technical report was prepared by Softech, Inc., Waltham, MA and covers work performed under Contract F33615-76-C-1192 during the period September through December 1976. The work was funded under Project 2003, Task 05, Work Unit 14 by the Air Force Avionics Laboratory, Wright-Patterson AFB, Ohio 45433. The Air Force contract monitors were Mark A. Pitts and Ronald Szkody.

Significant contributors to this report include C. Hitchon, D. Pfaw, and M. Willoughby.

| | |
|---------------------------------|--|
| ACCESSION for | |
| NTS | <input checked="" type="checkbox"/> File Section |
| DOC | <input type="checkbox"/> D II Section |
| SEARCHED | |
| INDEXED | |
| FILED | |
| BY | |
| DISTRIBUTION/AVAILABILITY CODES | |
| SPECIAL | |
| P1 | |

TABLE OF CONTENTS

| <u>Section</u> | | <u>Page</u> |
|----------------|--|-------------|
| I | BUILDING AN EXECUTIVE | 1 |
| 1.1 | Construction of Baseline Executives | 1 |
| 1.1.1 | File Name Conventions | 2 |
| 1.1.2 | Submit Files | 3 |
| 1.1.3 | Executive Construction | 4 |
| 1.2 | Construction of the Directed Flowgraph Model | 5 |
| 1.2.1 | Approach | 5 |
| 1.2.2 | Reduction of DFG Complexity | 6 |
| 1.2.2.1 | Preemption Reduction | 7 |
| 1.2.2.2 | Combining Tasks | 12 |
| 1.2.2.3 | Serializers | 14 |
| 1.2.2.4 | Complex DFG Constructs | 15 |
| 1.2.2.5 | Partitioning | 19 |
| 1.2.2.6 | Other Possible Simplifications | 19 |
| 1.2.3 | Final DFG Model | 20 |
| 1.2.3.1 | Key to DFG Models | 22 |
| 1.2.3.2 | Processor 0 DFG Model | 26 |
| 1.2.3.3 | Processor 1 DFG Model | 29 |
| 1.2.3.4 | Correspondence of DFG Model to Formal DFG | 30 |
| II | SYSTEM PARAMETERS | 32 |
| 2.1 | Processor 0 System Parameters | 33 |
| 2.2 | Processor 1 System Parameters | 36 |
| III | TUNING THE EXECUTIVE | 39 |
| 3.1 | Baseline Executive Statistics | 40 |
| 3.1.1 | ITC Baseline Statistics | 42 |
| 3.1.1.1 | Approach | 42 |
| 3.1.1.2 | Definition of ITC Activities | 42 |
| 3.1.1.3 | ITC Baseline Detailed Timing Statistics | 44 |
| 3.1.1.4 | ITC Baseline Detailed Space Statistics | 48 |
| 3.1.1.5 | ITC Sensitivity Analysis | 52 |
| 3.1.2 | TIM Baseline Statistics | 53 |
| 3.1.2.1 | Approach | 53 |
| 3.1.2.2 | Definition of TIM Activities | 54 |
| 3.1.2.3 | TIM Baseline Detailed Timing Statistics | 55 |
| 3.1.2.4 | TIM Baseline Detailed Space Statistics | 57 |
| 3.1.2.5 | Sensitivity Analysis | 58 |

TABLE OF CONTENTS (Continued)

| <u>Section</u> | | <u>Page</u> |
|----------------|--|-------------|
| III | TUNING THE EXECUTIVE (Continued) | |
| 3.1 | Baseline Executive Statistics (Continued) | |
| 3.1.3 | DAC Baseline Statistics | 59 |
| 3.1.3.1 | Approach | 59 |
| 3.1.3.2 | Definition of DAC Activities | 59 |
| 3.1.3.3 | DAC Baseline Detailed Timing Statistics | 61 |
| 3.1.3.4 | DAC Baseline Detailed Space Statistics | 63 |
| 3.1.3.5 | DAC Sensitivity Analysis | 65 |
| 3.1.4 | SCH Baseline Statistics | 66 |
| 3.1.4.1 | Approach | 66 |
| 3.1.4.2 | Definition of Activities | 67 |
| 3.1.4.3 | SCH Baseline Detailed Timing Statistics | 68 |
| 3.1.4.4 | SCH Baseline Detailed Space Statistics | 69 |
| 3.1.4.5 | SCH Sensitivity Analysis | 70 |
| 3.1.5 | DTF Baseline Statistics | 71 |
| 3.1.5.1 | Approach | 71 |
| 3.1.5.2 | Definition of DTF Activities | 72 |
| 3.1.5.3 | DTF Baseline Detailed Timing Statistics | 73 |
| 3.1.5.4 | DTF Baseline Detailed Space Statistics | 75 |
| 3.1.5.5 | Sensitivity Analysis | 76 |
| 3.1.6 | MSM Baseline Statistics | 77 |
| 3.1.6.1 | Approach | 77 |
| 3.1.6.2 | Definition of MSM Activities | 77 |
| 3.1.6.3 | MSM Baseline Detailed Timing Statistics | 78 |
| 3.1.6.4 | MSM Baseline Detailed Space Statistics | 79 |
| 3.1.6.5 | Sensitivity Analysis | 79 |
| 3.1.7 | DSP Baseline Statistics | 80 |
| 3.1.7.1 | Approach | 80 |
| 3.1.7.2 | Definition of DSP Activities | 81 |
| 3.1.7.3 | DSP Baseline Detailed Timing Statistics | 82 |
| 3.1.7.4 | DSP Baseline Detailed Space Statistics | 84 |
| 3.1.7.5 | DSP Sensitivity Analysis | 85 |

TABLE OF CONTENTS (Continued)

| <u>Section</u> | | <u>Page</u> |
|----------------|--|-------------|
| III | TUNING THE EXECUTIVE (Continued) | |
| 3.1 | Baseline Executive Statistics (Continued) | 85 |
| 3.1.8 | SSM Baseline Statistics | 85 |
| 3.1.9 | MPL Baseline Statistics | 85 |
| 3.1.10 | Bus Traffic | 85 |
| 3.1.10.1 | Processor 0 | 86 |
| 3.1.10.2 | Processor 1 | 86 |
| 3.1.10.3 | Interprocessor Bus Traffic | 86 |
| 3.2 | Tuning for HOL Inefficiencies | 87 |
| 3.2.1 | Overview | 87 |
| 3.2.2 | Examples | 87 |
| 3.2.2.1 | CLOCKQ | 87 |
| 3.2.2.2 | SEND | 88 |
| 3.2.2.3 | QUEUE and DQUEUE | 88 |
| 3.2.3 | Summary | 89 |
| 3.3 | Tuning by Reducing Generality | 89 |
| 3.3.1 | Overview | 89 |
| 3.3.2 | Examples | 89 |
| 3.3.2.1 | CLOCKQ | 89 |
| 3.3.2.2 | SEND | 89 |
| 3.3.2.3 | QUEUE and DQUEUE | 90 |
| 3.3.3 | Summary | 90 |
| 3.4 | Tuning for HOL Language Deficiencies | 91 |
| 3.4.1 | Overview | 91 |
| 3.4.2 | Examples | 91 |
| 3.4.2.1 | CLOCKQ | 91 |
| 3.4.2.2 | SEND | 92 |
| 3.4.2.3 | QUEUE and DQUEUE | 92 |
| 3.4.3 | Summary | 93 |
| 3.5 | Final Tuning | 94 |
| 3.5.1 | Overview | 94 |
| 3.5.2 | Final Tuning Description | 95 |
| 3.6 | Final Tuned Executive | 97 |
| 3.6.1 | ITC Final Tuned Statistics | 99 |
| 3.6.1.1 | ITC Tuning Approach | 99 |
| 3.6.1.2 | Functional Differences with Baseline | 100 |
| 3.6.1.3 | ITC Final Tuned Detailed Timing Statistics | 101 |
| 3.6.1.4 | ITC Final Tuned Detailed Space Requirements | 103 |
| 3.6.1.5 | ITC Sensitivity Analysis | 105 |

TABLE OF CONTENTS (Continued)

| <u>Section</u> | | <u>Page</u> |
|----------------|---|-------------|
| III | TUNING THE EXECUTIVE (Continued) | |
| | 3.6 Final Tuned Executive (Continued) | |
| | 3.6.2 TIM Final Tuned Statistics | 106 |
| | 3.6.2.1 TIM Timing Approach | 106 |
| | 3.6.2.2 Functional Differences with Baseline | 107 |
| | 3.6.2.3 TIM Final Tuned Detailed Timing Statistics | 107 |
| | 3.6.2.4 TIM Final Tuned Detailed Space Statistics | 108 |
| | 3.6.2.5 TIM Sensitivity Analysis | 109 |
| | 3.6.3 DAC Final Tuned Statistics | 109 |
| | 3.6.3.1 DAC Tuning Approach | 109 |
| | 3.6.3.2 Functional Differences with Baseline | 110 |
| | 3.6.3.3 DAC Final Tuned Detailed Timing Statistics | 110 |
| | 3.6.3.4 DAC Final Tuned Detailed Space Statistics | 111 |
| | 3.6.3.5 DAC Sensitivity Analysis | 112 |
| | 3.6.4 SCH Final Tuned Statistics | 113 |
| | 3.6.4.1 SCH Tuning Approach | 113 |
| | 3.6.4.2 Functional Differences | 113 |
| | 3.6.4.3 SCH Final Tuned Detailed Timing Statistics | 114 |
| | 3.6.4.4 SCH Final Tuned Detailed Space Statistics | 115 |
| | 3.6.4.5 SCH Sensitivity Analysis | 116 |
| | 3.6.5 DTF Final Tuned Statistics | 116 |
| | 3.6.5.1 Tuning Approach | 117 |
| | 3.6.5.2 Functional Differences with Baseline | 117 |
| | 3.6.5.3 DTF Final Tuned Detailed Timing Statistics | 118 |
| | 3.6.5.4 DTF Final Tuned Detailed Space Requirements | 120 |
| | 3.6.5.5 DTF Sensitivity Analysis | 122 |
| | 3.6.6 MSM Final Tuned Statistics | 123 |
| | 3.6.6.1 Functional Differences with Baseline | 123 |
| | 3.6.6.2 Tuning Approach | 123 |
| | 3.6.6.3 MSM Final Tuned Detailed Timing Statistics | 124 |
| | 3.6.6.4 MSM Final Tuned Detailed Space Statistics | 124 |
| | 3.6.6.5 Sensitivity Analysis | 124 |

TABLE OF CONTENTS (Continued)

| <u>Section</u> | | <u>Page</u> |
|----------------|--|-------------|
| III | TUNING THE EXECUTIVE (Continued) | |
| 3.6 | Final Tuned Executive (Continued) | |
| 3.6.7 | DSP Final Tuned Statistics | 125 |
| 3.6.7.1 | DSP Tuning Approach | 125 |
| 3.6.7.2 | Functional Differences | 126 |
| 3.6.7.3 | DSP Final Tuned Timing Statistics | 126 |
| 3.6.7.4 | DSP Final Tuned Detailed Space Statistics | 128 |
| 3.6.7.5 | DSP Sensitivity Analysis | 129 |
| 3.6.8 | SSM Final Tuning Statistics | 129 |
| 3.6.9 | MPL Final Tuning Statistics | 129 |
| 3.6.10 | Final Tuned Bus Traffic | 129 |
| 3.7 | Conclusions | 129 |
| 3.7.1 | Statistical Reduction Allocated by Function Cluster | 129 |
| 3.7.2 | Statistical Reduction Allocated by Tuning Method | 134 |
| Appendix A | EXECUTIVE TUNING EXAMPLES | A-1 |
| Appendix B | TUNING THE DAIS MISSION DFG | B-1 |
| Appendix C | ASSIGNMENT OF BUS SUBADDRESSES | C-1 |

LIST OF ILLUSTRATIONS

| <u>Figure No.</u> | | <u>Page</u> |
|-------------------|--|-------------|
| 1a | FILE NAME CONVENTIONS | 2 |
| 1b | EFFECT OF PREEMPTION ON DATA ACCESS CONTROL | 10 |
| 2 | BASELINE EXECUTIVE TIMING STATISTICS | 41 |
| 3 | BASELINE EXECUTIVE SPACE STATISTICS | 41 |
| 4 | FINAL TUNED EXECUTIVE TIMING STATISTICS | 98 |
| 5 | FINAL TUNED EXECUTIVE SPACE STATISTICS | 98 |
| 6 | PROCESSOR 0 TIME OVERHEAD | 130 |
| 7 | PROCESSOR 0 SPACE OVERHEAD | 131 |
| 8 | PROCESSOR 1 TIME OVERHEAD | 132 |
| 9 | PROCESSOR 1 SPACE OVERHEAD | 133 |

Section I
BUILDING AN EXECUTIVE

Building a baseline (untuned) OSC executive for a particular mission is a two-step process. First, the formal Directed Flowgraph (DFG) which specifies the mission must be studied, simplified if possible, and converted into an executive DFG model which can be translated directly into an executive data set. This process is described in detail in Section 1.2.

In the second step, the resulting executive data set is edited into the appropriate COPY and COMPOOL source files as preset data. The baseline mission executive is then created by performing the proper sequence of compilations and link edits. This process is described in detail in Section 1.1.

The resulting executive is fully functional but untuned. The processes of tuning and analysis of the results are described in detail in Section 3.

1.1 Construction of Baseline Executives

The baseline executives for processor 0 and processor 1 can be created from files residing on PPN [1111, 351] of the DAIS PDP-10. Special file naming conventions have been used to aid in the identification of files and construction of the executives. These conventions will be described in the first subsection.

The creation of an executive is a three-step process. First, all the required source files are moved to a separate PPN. Then they are compiled, assembled and reformatted. Finally, they are linked to create the desired executive. The executives are named CSC0.LDA for processor 0 and OSC1.LDA for processor 1. Each of these processes is aided by a set of submit files on PPN[1111, 420]. These files are prepared batch jobs to make the process easier and more reliable. The second subsection describes the necessary submit files, the final subsection describes how they are used to create the executive.

1.1.1 File Name Conventions

Each file name consists of two parts separated by a dot. The name before the dot is used to indicate the type of information it contains; that is, it indicates whether the file contains executive procedures, data declarations, procedure declarations or initialization procedures. The name after the dot, the extension, indicates the executives the file is used to build, and whether it is a J73I program, DAIS assembly language program, copy file, COMPOOL source or preprocessed COMPOOL. A summary of the conventions is given in Figure 1a. The use of a "?" within a name indicates that the appropriate letter is substituted for each "?" to name the requested file. For example, the designation ???IPD indicates that names such as ITCIPD, DACIPD, and MSMIPD are valid.

| File Type | Common Name | Processor 0 Name | Processor 1 Name |
|---|-------------|------------------|------------------|
| J73I programs | .J73 | .J70 | .J71 |
| DAIS Assembly language programs | .DAL | .DA0 | .DA1 |
| copy files | .CPY | .CP0 | .CP1 |
| COMPOOL source files | .CPS | .CS0 | .CS1 |
| Preprocessed COMPOOL (one for each CCOMPOOL file) | .CMP | .CMP | .CMP |
| Executive procedures | ???PRC. | ???PRC. | ???PRC. |
| Initialization procedures | ???INT. | ???INT. | ???INT. |
| Data declarations | ???DCL. | ???DCL. | ???DCL. |
| Separate copy of initial data | ???IDT. | ???IDT. | ???IDT. |
| External procedure declarations | ???EPD. | ???EPD. | ???EPD. |
| Internal procedure declarations | ???IPD. | ???IPD. | ???IPD. |

Figure 1a. FILE NAME CONVENTIONS

1. 1. 2 Submit Files

A set of submit files resides on PPN[1111,420] to assist in constructing an executive. The following list names and describes each of them.

- | | | |
|------------|---|--|
| MVOSC0.CTL | - | copy all files in PPN[1111,351] needed to create the processor 0 executive. Rename all .CP0, .CS0, .J70 and .DA0 extensions to .CPY, .CPS, .J73 and .DAL. |
| MVOSC1.CTL | - | copy all files in PPN[1111,351] needed to create the processor 1 executive. Rename all .CP1, .CS1, .J71 and .DA1 extensions to .CPY, .CPS, .J73 and .DAL. |
| PRC.CTL | - | compile all executive J73 programs. |
| PD.CTL | - | compile all procedure declaration COMPOOL files. |
| CMP.CTL | - | compile all COMPOOL files that are not procedure declarations. |
| FMT.CTL | - | reformat all executive .REL files into .DAT files |
| ASSMBL.CTL | - | assemble all executive DAIS Assembly Language files. |
| LINK0.CTL | - | create the processor 0 executive OSC0.LDA. |
| LINK1.CTL | - | create the processor 1 executive OSC1.LDA. |

1. 1. 3 Executive Construction

This description will assume the processor 0 executive is being created. The processor 1 executive is created in the same way, with the appropriate submit files substituted. One executive should be completed before the next is begun. It is also important that the commands be performed in the order listed to insure that the proper files are used. It is also important that each command complete before the next one begins.

The first step in building the executive is to copy the appropriate files to a different PPN. This can be done with submit files residing on [1111,420]. These submit files will copy to whatever PPN the submit file is on. The following command will copy the necessary files:

. SUBMIT MVOSC0 (MVOSC1)

All of the required source files have now been copied. The COMPOOLS and programs must be compiled and assembled next.

. SUBMIT CMP
. SUBMIT PD
. SUBMIT PRC
. SUBMIT ASSMBL

Then the files are reformatted and linked to create the executive.

. SUBMIT FMT
. SUBMIT LINK0 (LINK1)

The resulting executive will be named OSC0.LDA (OSC1.LDA).

1.2 Construction of the Directed Flowgraph Model

1.2.1 Approach

The process of constructing an OSC DFG model begins when the mission planner has produced a complete specification of the mission's executive requirements in the form of a formal DFG together with supplementary information. The formal DFG precisely depicts all data and control relationships between the mission's tasks, sources, and sinks. Supplementary information provides task execution rates and times, data link descriptions, memory and response requirements, and any other quantitative aspects of system performance or resource requirements which may influence construction of the executive DFG model.

Each mission task depicted on the formal DFG is considered as an indivisible unit. The internal operations of these tasks are not specified in the formal DFG and are, in fact, only pertinent in their effects on the supplementary quantitative information.

The job of the executive builder is to construct an OSC executive which accurately models the formal DFG (in that the data and control flow will be as specified) and which also meets the quantitative requirements (such as execution rates, response times, etc.) on the target hardware configuration. More specifically, since the OSC executive is a table-driven DFG interpreter, the job of the executive builder is to construct a set of tables which will drive the executive correctly and meet the quantitative requirements.

One approach to this problem is to perform a one-to-one mapping of the formal DFG's links, nodes, sources, etc. into the corresponding models supported by the OSC executive. It is quite possible that such a "brute force" translation of the formal DFG into an executive data set could be performed automatically by a translator program driven by a formalized linguistic representation of the DFG. This approach has a serious drawback. A direct translation of a relatively complex DFG may result in an executive which is so large and slow that space and execution time requirements of the mission's tasks cannot be met. Moreover, since such an automatic translator does not exist, a tedious, error-prone hand translation must be performed.

For these reasons, the executive builder may be forced to reduce the complexity of the DFG prior to final translation into an executive data set. The need to reduce complexity, however, does not negate the usefulness of an automatic translator. Even executive data sets produced by the translator which turn out to be too inefficient, can be made useful tools for debugging the DFG by using task stubs and running virtual clocks at a slower rate.

1.2.2 Reduction of DFG Complexity

The reduction of the complexity of a detailed formal DFG takes place in successive stages. At each stage, a less complex model of the DFG results. Each successive model must meet the data and control constraints of the original formal DFG while also more closely approaching a final DFG model whose run time overhead is low enough to leave sufficient resources for the mission tasks. The process of complexity reduction need not stop when this point is reached. Indeed, continuing will further reduce the bulk of error prone translation required and also reduce executive overhead to allow for future expansion of mission resource requirements.

The simplifications achieved at each stage of DFG complexity reduction may result from any of a number of DFG tuning methods (described below). Specific examples of each tuning method as applied to the DAIS mission DFG are provided along with the method descriptions. Details of the final tuned version of the DAIS DFG are presented in Section 1.2.3. The notation used in the final DFG is not strictly formal DFG notation but is an adaptation biased toward the actual OSC executive modeling of DFG objects. A key to this notation is also presented in Section 1.2.3. Although only the final result of the DFG tuning process is diagrammed in section 1.2.3, several intermediate tuned versions were sketched during the tuning process. These intermediate versions are included in Appendix B.

The DFG complexity reduction process (i. e., DFG tuning) is distinct from tuning of the executive programs and their data structures.

However, there is an important interaction to consider. Tuning of the DFG may result in a model which requires considerably less than the full functionality provided by the baseline executive. In anticipation of this situation the functional features of the OSC executive were designed and implemented in a modular fashion to make their deletion a simple matter of excluding certain procedures or data structures from the executive built. It is worth noting that tuning of the formal DAIS DFG resulted in a significant reduction in the various executive functions required. In fact, further tuning of the DFG was deliberately avoided in order to retain examples of all important executive capabilities.

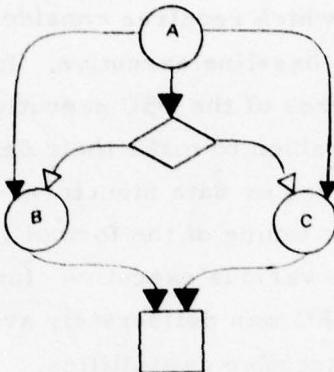
1.2.2.1 Preemption Reduction

The execution of two tasks, A and B, is said to be interleaved if part or all of task A is executed after task B begins execution but before B completes or vice versa. The execution of tasks A and B is further said to be concurrent if at any time both A and B are in execution. Concurrent execution can occur only in configurations containing more than one processor. Examples are multiprocessor systems where memory is shared, and federated processor systems where a data transport resource is shared. In a uniprocessor system, only interleaved execution of tasks is possible. In particular, each processor in a federated configuration is a uniprocessor in its own context. In a uniprocessor system, interleaved execution occurs when one task preempts another, i.e., when the execution of one task is temporarily suspended to allow execution of a more urgent task.

A formal DFG may impose many constraints on task preemption. For example, in the DFG below, task B must complete execution before task A can be executed.



In the next DFG, execution of tasks A, B, and C will be mutually exclusive.



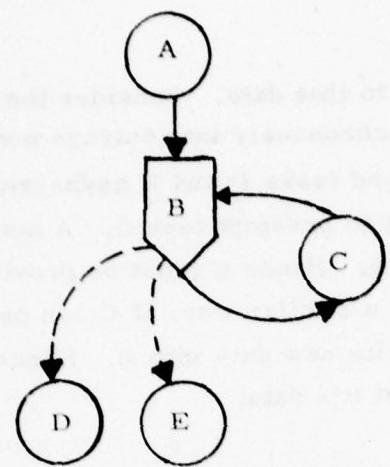
For other tasks, the DFG may impose few constraints or none at all. The purpose of the formal DFG is to specify exactly those data and control constraints that are required for proper system operation and no more. Since all data and control relationships among the tasks are specified on the formal DFG, any task execution policy satisfying the DFG constraints will perform correctly.

The executive builder's options in construction are proportional to the level of detail on the formal DFG specification. The further each task is broken down into smaller tasks, the easier it is to isolate the sources of contention for resources.

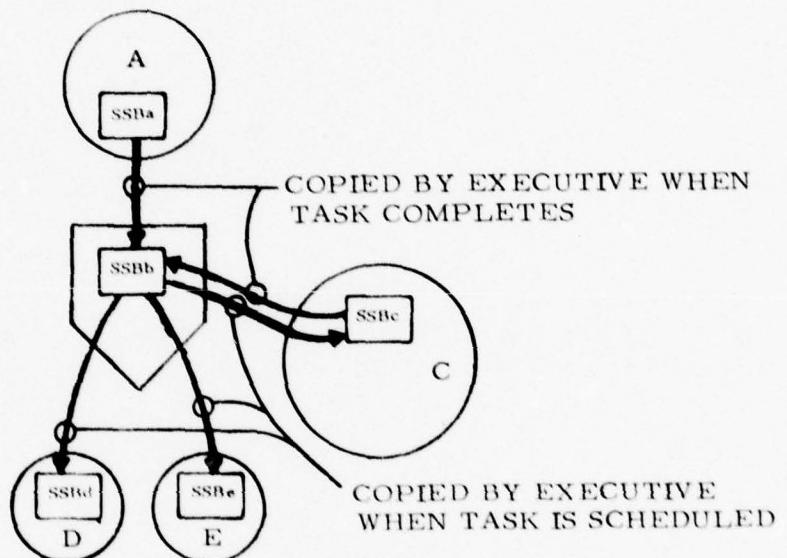
The purpose of permitting preemption of one task by another within a uniprocessor system is only to satisfy response requirements. That is, a task with short response requirements may be required to run at a time when another task is already running. The task already in execution may take so long to complete that the other task's response requirement cannot be met if the executing task is allowed to run to completion.

The reduction or elimination of preemption requirements is the single most effective method for reducing the complexity of the DFG model. The reason for this is that tasks which cannot preempt one another can access the same global data without a contention problem (e.g., one task reading the data while the other is writing it). If tasks which must preempt one another also contend for global data, then the

executive must coordinate access to that data. Consider the example in Figure 1. Task A writes asynchronously into storage node B, task C synchronously updates B, and tasks D and E asynchronously read B. Now suppose task A is allowed to preempt task C. A may then change the contents of B while C is running. Hence C must be provided with a separate copy of the data in B. In a similar way, if C can preempt D, then while D is running, C may write new data into B. Hence D must be provided with a separate copy of B's data.



FORMAL DFG REPRESENTATION



STATIC BLOCK EXECUTIVE IMPLEMENTATION
SUPPORTING UNLIMITED PREEMPTION

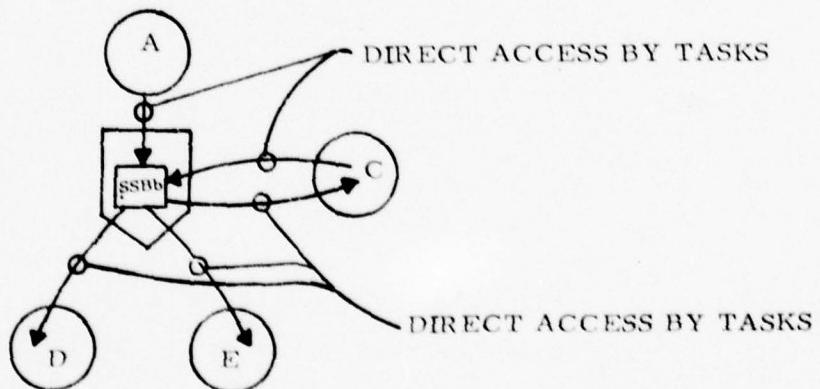


Figure 1b. EFFECT OF PREEMPTION ON DATA ACCESS CONTROL

The OSC executive provides the facilities of the DAC (Data Access Control) cluster to manage such situations. There are basically two ways of handling these problems with DAC, one in which the data is statically allocated (fixed locations for data buffers) and one in which the data space is dynamically allocated (locations for data buffers are determined at run time). Which solution is better depends upon the length of the data and the execution rates of the accessing tasks. In the static method the "current" data for B is always contained in a global statically allocated storage block (SSB). Each time a task which reads B is scheduled for execution, the executive copies the data in B to a local SSB which the task accesses instead of the global copy. Each task which writes into B writes instead into its own local copy of B and the executive copies the data into the global SSB when the task completes (see Figure 1). In the dynamic method, static storage blocks (SSBs) are replaced by dynamic storage blocks (DSBs) and only pointers to the DSBs are copied by the executive.

Clearly, considerable executive overhead may be required to perform such control of data access. On the other hand, if tasks A, C, D and E in Figure 1 are not allowed to preempt one another, then each task will have exclusive access to B while it executes. In this case, no contention problem exists, and each task may directly reference the single global block B.

Another problem associated with preemption is that of application programs which are shared by two or more application tasks. If two tasks which can preempt one another invoke the same subroutine, then that subroutine must be reentrant. Reentrancy is usually obtained at some cost in efficiency, and in the case of J73I an error prone program controlled stack management.

In summary, the key factors determining the need for preemption are the response required for task execution and the maximum individual task execution times. In general preemption can be avoided when the maximum task execution time is small relative to most severe response requirements. For this reason, it is important that the mission planner

specify the system with the most detailed DFG practical. In the DAIS mission DFG, the maximum exemption time of a single task was approximately 6 milliseconds while the tightest response requirement was approximately 30 milliseconds. Even under maximum load conditions, it was possible to meet the response requirements without allowing any preemption. However, in order to demonstrate the executive handling of preemption, the longest task combination, AT20 on the final DFG with execution time 8 milliseconds, was made preemptable. As a consequence, data selector 21 and the link data coming into AT20 had to be put under executive management.

1.2.2.2 Combining Tasks

Another important means of simplifying the DFG is to combine tasks on the formal DFG into larger tasks. This is accomplished by writing a skeletal master task which simply calls each task in the combination as a subroutine. Since subtasks within such a task combination are executed sequentially, there is no mutual preemption. There are several benefits to be gained by combining tasks:

- Executive scheduling overhead is reduced since one scheduling of the combined task is equivalent to scheduling all subtasks.
- Executive table space required for the combined task is the same as the space otherwise required for one subtask.
- The source/sink requirements of each subtask are combined resulting in fewer calls to DTF, a reduction in the number of access controllers required and batching of I/O operations.
- Control signals which individually activate each subtask are combined into a single control signal.

The benefits of combining tasks must be realized while adhering to the DFG specification. Tasks may be combined without deviating from DTF requirements if they are executed under the same conditions, for example, tasks which are controlled by the same clock or control link (through an identity). In such task combinations, the skeletal master task simply calls each subtask in any order. Tasks which are connected to one another by simple data links may be combined. In these task com-

binations, the skeletal master task calls each subtask after all the subtasks providing input links to that subtask have been executed. Calls to subtasks in the master skeletal task are carefully ordered to reflect link imposed execution order. Finally, tasks interconnected by control selectors may be combined. The control selector nodes which connect the tasks are implemented in the skeletal master task as if-then-else statements which call each subtask when the corresponding control selector predicate indicates.

In many task combinations created for the DAIS DFG, subtasks which are conditionally executed produce output to devices. A new software signal capability (via procedure SIGAC) was added to allow the skeletal master tasks to conditionally signal that one or more output operations be performed when the task combination completes.

Large combinations of tasks can result in an increase in the maximum task execution time to a point where some task combinations must be made preemptable in order to meet response requirements. This can result in an increase in executive data access control overhead greater than the overhead saved by combining the tasks. Hence, task combinations which complicate meeting response requirements should be avoided. In the DAIS mission DFG, such combinations were avoided except in the case of AT20. AT20 is the longest executing task combination (8 milliseconds) and was made preemptable for the purpose of demonstrating the executive's preemption capability.

Some examples of task combinations in the tuned DAIS DFG are AT01 which combines most of the tasks and control selectors involved in handling asynchronous pilot inputs, and AT18 which combines several tasks which must run at the 8/sec rate. Complete details of the tuned DAIS DFG are presented in Section 1.2.3. Details of the flow of control within each task combination's master skeletal task are specified in the J73I program APPRC.J70 for processor 0 and in APPRC.J71 for processor 1.

1.2.2.3 Serializers

The serializer nodes drawn on a DFG may have quite complex implications. In the worst case, it implies a number of input links containing data which must be queued by the executive as they become enabled and then serially removed from the queue and copied to the output link. However, in actual practice, this full functionality of the serializer may not be required.

In the simplest case a serializer may imply enabling a particular link whenever any one of a number of mutually exclusive links is enabled. In such cases no queueing is required and each input link to the serializer may be replaced by the single output link. Examples of such serializers on the DAIS DFG abound. The serializers which control the iterative tasks T53 and T55 on page 3B of the DFG are good examples as are the serializers on the same page which accept data from each loop output.

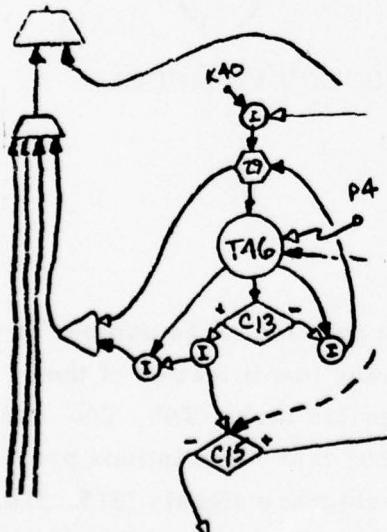
A slightly more complicated case is a control serializer with inputs which are not mutually exclusive. Again executive queueing is easily avoided by allowing the wait count for the task connected to the output link to be decremented below zero. When the task completes it is rescheduled if its recomputed wait count is still less than or equal to zero. On the DAIS DFG examples of such tasks are T36 and T22 (AT36 and AT22 on the tuned DFG).

Finally, there is a more difficult type of data serializer where again mutual exclusion is not obvious from the DFG. Here it is possible to guarantee serialization by including the serializer's output task in a separate task combination with each input task to the serializer and disallowing mutual preemption of these task combinations. It is important to realize that this method neither implies that multiple copies of the output task are required nor that the task must be reentrant. Examples of this type of serializer implementation in the tuned DFG are tasks AT01, AT08, and AT09 which all may call the same subtask (T08) and cannot preempt one another. In some cases, the inclusion of a subtask in two or more task combinations has forced duplication of its output devices so that a different data area can be used for each instance.

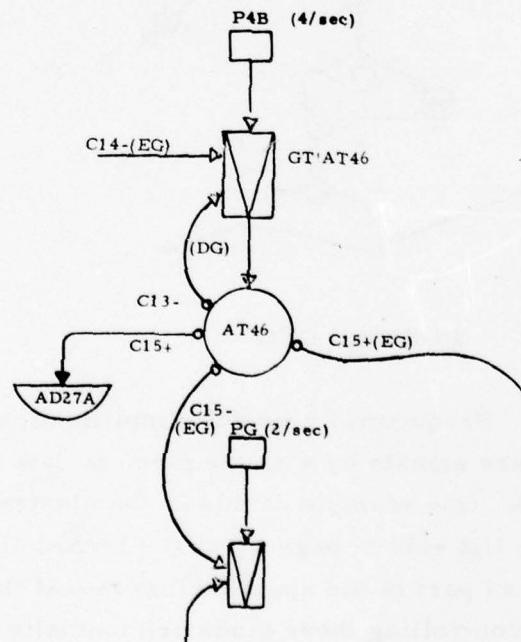
1.2.2.4 Complex DFG Constructs

In some cases complex appearing DFG constructs yield to quite simple executive implementation. That is, in some cases, the OSC executive can model a complex combination of nodes without modeling each individual node. Good examples of this type of simplification are the eight clock controlled loops on page 3B of the formal DAIS DFG specification. These loops were reduced to simple combinations of gates and tasks in the tuned DFG. The control selectors which monitor each loop have been subsumed into each loop task and are manifested as software signals which enable and disable the appropriate gates.

The technique of combining control selectors into a combination with the task which produces the controlling data and allowing the skeletal master task for this combination to produce the appropriate software signals was used throughout the tuned DFG to eliminate the unnecessary overhead of treating each control selector as a separate node.

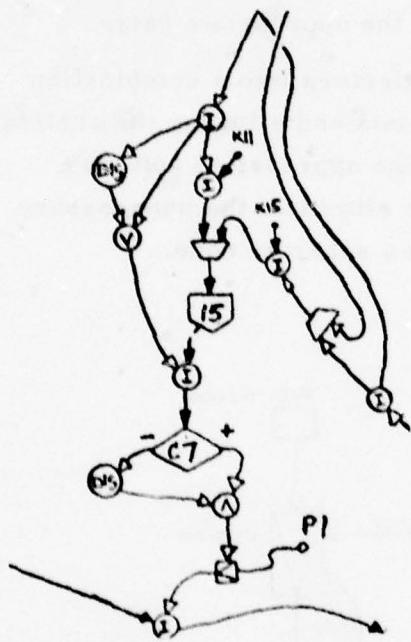


FORMAL DFG



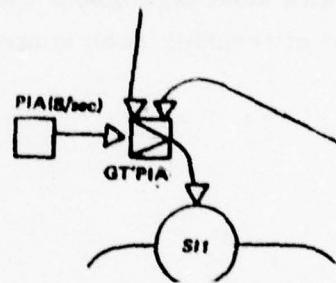
TUNED DFG MODEL

Sometimes representation of a control structure in terms of DFG symbols is rather awkward. A case in point is the group of nodes which ultimately controls the gate on clock signal P1 in the lower left corner of page 2A of the formal DFG. What is specified is much simpler than it appears, namely control selector C3, C4 and C10 all control the gate. The executive implementation reduces to a simple enabling or disabling of the gate by software signals generated in the skeletal master tasks for the task combinations AT01 and AT09 on the tuned DFG.



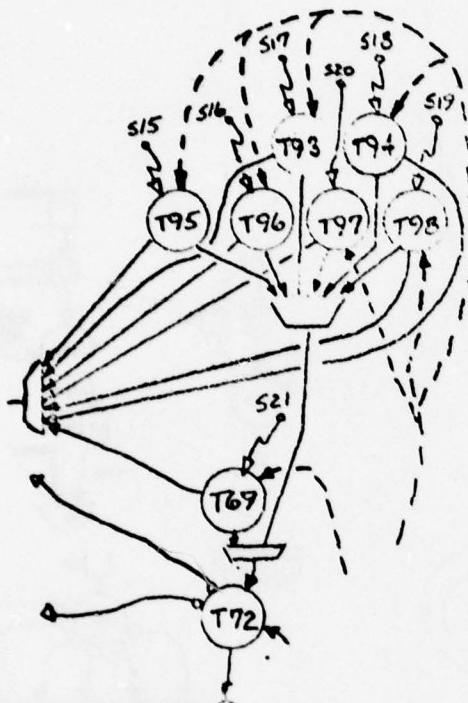
FORMAL DFG

Frequently, a useful simplification is to replace a large number of software signals by a single piece of data containing identification of the signal. One example of this is the cluster of signaled tasks (T95, T96, etc.) on the left side of page 4A of the formal DFG. The task descriptions provided as part of the specification reveal that the software signals (S15, S16, etc.) controlling these tasks are mutually exclusive. Because of this it is possible to replace all those signals with a single signal and a data word which specifies which signal has occurred.

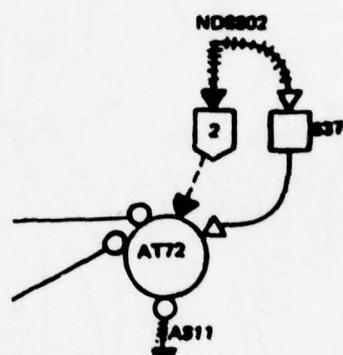


TUNED DFG MODEL.

The associated tasks can be bound into a single task combination signaled by the new signal. The skeletal task for the combination can use the data word as an index in a SWITCH statement to dispatch control to the appropriate subtask.

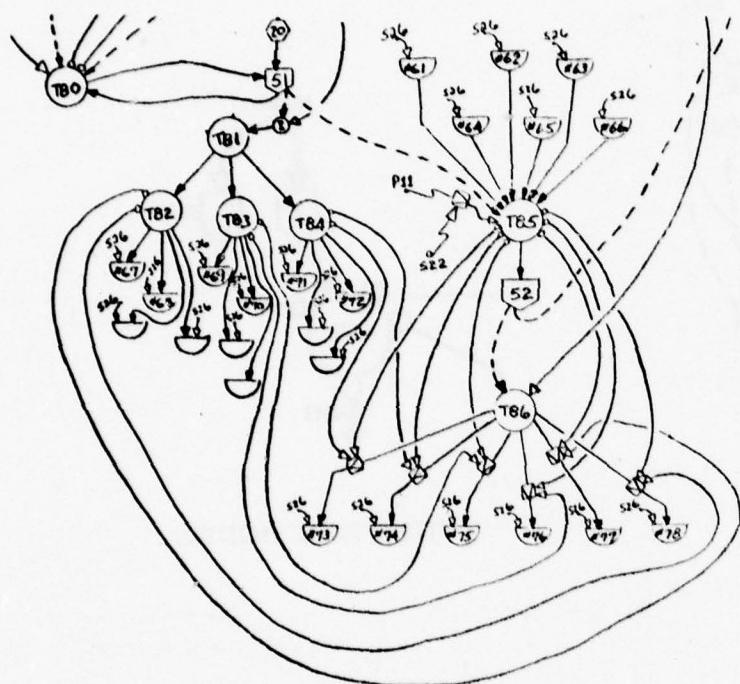


FORMAL DFG

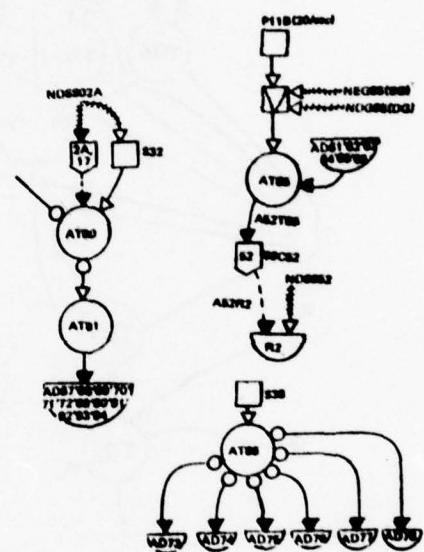


TUNED DFG MODEL

Another example of a simplifying signal/data trade involves the tasks at the top of page 4B of the formal DFG. Here, many signalled control links select an output device for T86. Instead, the control signals can be combined into a simple bit mask which is updated by the control tasks and read by T86 to select the proper output device.



FORMAL DFG



TUNED DFG MODEL

1.2.2.5 Partitioning

Many factors must be considered in partitioning the DFG among processors. The fact that the formal DFG was designed for a four processor system but had to be partitioned for a two processor system made careful consideration of these factors critical. These factors include memory usage balance (for both executive and applications), functional isolation to avoid total system failure in the event of a single processor failure, processor time requirements, transfer rates along inter-processor links, distribution of source sink load, and allocation of subaddresses. Generous detail in the formal DFG is an important aid to effective partitioning.

In partitioning the DAIS DFG all these factors were considered. The result consists of a well balanced two processor partitioning of an application intended for four processors. Most of the inter-processor communication is from processor 0 to processor 1. Data is sent from processor 0 to processor 1 each time a data selector which is asynchronously accessed (at a high rate) by processor 1 is updated by processor 0. Processor 0 also requires asynchronous access to some data selectors in processor 1, namely 43, 44 and 52. These accesses occur at a low rate while updating in processor 1 occurs at a high rate. Hence, rather than sending this data via DTF each time it is updated, processor 0 sends a request for the data (requiring transmission of one data word) when it is needed. Processor 1 then responds by sending back the requested data.

1.2.2.6 Other Possible Simplifications

Because of the desire to demonstrate various executive features, not all possible simplifications were applied to the formal DAIS DFG. In particular, more tasks could have been combined into larger tasks. For example, tasks running at one rate could be combined with tasks running at a slightly higher rate. Although such tasks would be executed at a higher rate than required, thus consuming more processor time, gains in reduced executive overhead obtained through elimination of tasks and clocks might more than compensate for this apparent inefficiency.

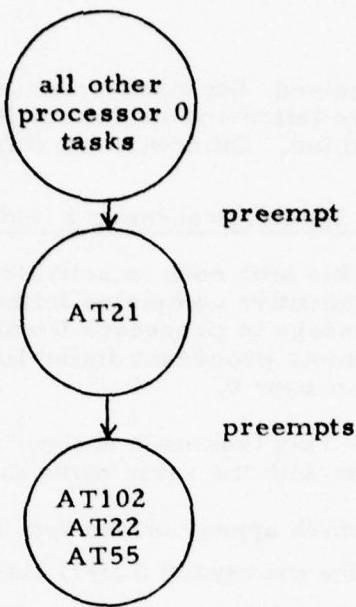
Tasks T89, T90, and T91 present an instance where a signal/data trade could have been made eliminating two task nodes, two clock pins, three signals and three gates. In this simplification the gating signals produced by T87 would be replaced by a data word which selects the appropriate subtask in a combination task including T89, T90, and T91.

1.2.3 Final DFG Model

The following subsections (1.2.3.1 - 1.2.3.4) present the results of the DFG complexity reduction process as applied to the DAIS mission DFG. The notation used is not the formal DFG notation but an adaptation of this notation which corresponds more closely to the OSC executive constructs which model a DFG. The meaning of each symbol used is specified in the subsection which follows.

Appendix B includes working DFG's which were drawn at various stages in the tuning process. The final tuned version which follows is the result of repeated application of all the tuning techniques described in the preceding sections. The constructs which are included in the final tuned version are only those which require active management by the OSC executive for correct DFG operation. Constructs not requiring executive management were omitted to simplify the diagram.

Although all of the tuning methods described were applied, two of them account for most of the simplification achieved. One is the combination of tasks which are activated under identical conditions into larger tasks. This tuning method is described in Section 1.2.2.2. The resulting task combinations created for the Dais mission DFG are listed in Section 1.2.3.4. The other frequently used tuning technique is limiting of intertask preemption. This technique is described in Section 1.2.2.1. Although the mission response requirements do not mandate any intertask preemption, preemption of the longest executing tasks was permitted in order to demonstrate the executive's full capabilities. The preemption structure for the processor 0 DFG is:



Tasks which cannot preempt one another can share access to data selector storage nodes without executive intervention. It is for this reason that only a few of these nodes appear in the final tuned DFG. The storage nodes which do appear are exactly those whose access must be managed by the executive to avoid contention among tasks which can preempt one another. It is also the reason that several tasks appear to have no data inputs and/or outputs (in particular task combinations AT55, AT68, AT81, and AT86).

Some of the task nodes in the final DFG do not appear in the original formal DFG. These nodes are special purpose nodes which aid the executive in initialization and failure detection. They are described in detail below.

Special Processor 0 Nodes

- 1) RFIN - this sink node when activated signals completion of initialization of both processors and causes the processor 0 executive to start its periodic clocks running.
- 2) RCLK - This sink node is activated periodically by a clock signal. It causes a special message, which is used to synchronize the real times clocks in both processors, to be sent to processor 1.
- 3) ATRPA - This task node is activated periodically by a clock signal. Each time it is activated, it checks a flag which is set if a message has been received from the other processor. If no message has been

received, the other processor is assumed to have failed and its failure link (RPFAIL) is enabled. Otherwise the flag is simply reset.

Special Processor 1 Nodes

- 1) RFIN - This sink node is activated when the processor 1 executive completes initialization. It sends a message to processor 0 which causes link RFIN (remote processor finish link) to be enabled in processor 0.
- 2) ATRPA - This task node has the same function as the node with the same name in processor 0.

The numbers which appear on the data links which are input to nodes D11 and D12 in the processor 0 DFG also have a special meaning. They indicate the number of times each link must be enabled before the terminating node is activated. In effect these links function as speed changes (discarders). This implementation of speed changers is more efficient for discarding ratios which are integral than the discarding node itself.

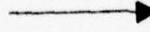
1.2.3.1 Key to DFG Models

Application Task



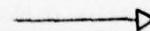
ATxxx is index of corresponding node.

Data Link



Nxx or Axx is the index of the DTF notification or DAC access controller associated with the link (if any).

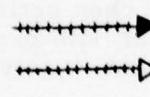
Control Link



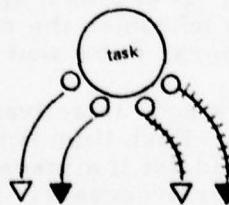
Asynchronous Access Link



Data and Control
link to and from
remote processor



Software signaled links
(link signaled by a task
via executive interface
procedures: SIGNLEVENT,
ENBLGATE, DSBLGATE,
or SIGAC).

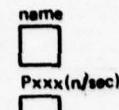


Pin (holds event time for the most recent enabling).

Clock Pin (frequency is given in parenthesis).

Gate

Task with serialized control (i.e., task runs once for each enabling of a, b, and c even if they occur simultaneously).



name is index of pin



Pxxx is index of pin

GT'xxx is index of gate

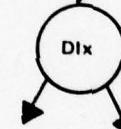


Simple Identity (control)



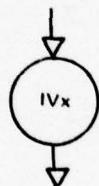
SIx is index of node

Data Identity



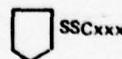
DIx is index of node

Inverter (changes enable to disable).



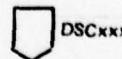
IVx is index of node

Data Selector Storage Node (static allocation)



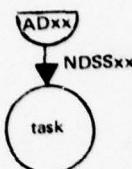
SSCxxxx is index of corresponding static storage controller.

Data Selector Storage Node (dynamic allocation)



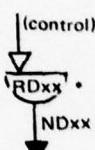
DSCxxxx is index of corresponding dynamic storage controller.

Source (for tasks). Data is read from the source when the task is ready to be scheduled. When notification is received that the data has arrived, the task is scheduled.



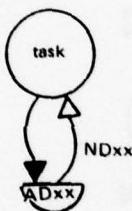
ADxx is the access-controller on the task node's begin access list which causes the source to be read. Dxx is the name(s) of the device(s) read, NDxx is the notification code provided by DTF when the data has been read.

Source (stand alone). Data is read from the device when the control link becomes enabled, notification is provided when the transfer is complete.



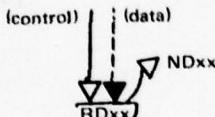
RDxx is the index of the source node (RL node), Dxx is the name(s) of the device(s) read, NDxx is the notification code provided by DTF when the data has been read.

Sink (for tasks). Data is written to the sink when the task completes. If a notification link is shown, then the task may not be scheduled for execution again until output is complete.



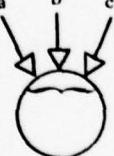
ADxx is the index of the access controller on the task node's end access list which causes the data to be written, Dxx is the device name(s), NDxx (optional) is the notification code provided by DTF when the data has been written.

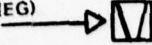
Sink (stand alone). Data (accessed via the asynchronous link) is written to the device(s) when the control link becomes enabled, notification is provided when the transfer is complete.

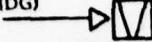


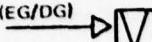
RDxx is the index of the sink node (RL node), Dxx is the device name(s), NDxx is the notification code provided by DTF when the transfer is complete.

Node which becomes active whenever a, b, or c is enabled. a, b, and c never occur simultaneously.

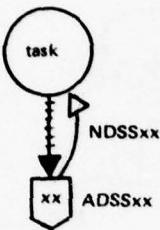


Link which enables a gate. (EG) 

Link which disables a gate. (DG) 

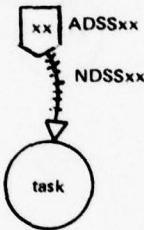
Link which can enable or disable a gate. (EG/DG) 

Link to remote processor which carries data selector data.



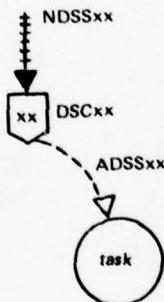
xx is name(s) of data selector node(s) whose data is sent to the remote processor. ADSSxx is the access controller in the task's end access list which causes the transfer, and NDSSxx is the notification code (if any) provided by DTF when the transfer is complete.

Asynchronous access link to data selector node in a remote processor. When task node becomes active, a message requesting the data is sent to the remote processor. When data is received from the remote processor, the task is scheduled for execution.



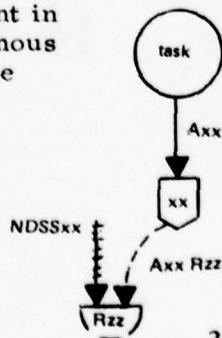
ADSSxx is the index of the access controller in the task node's begin access list which causes a message to be sent to the remote processor requesting the data in the data selector(s) xx. When the data is received DTF notifies the task node via the notification code NDSSxx.

Link which carries data from a remote data selector storage node asynchronously for asynchronous access.



NDSSxx is the notification code provided by DTF whenever data for the data selector(s) xx is received. DSCxx is the index of the dynamic storage controller for the data selector(s). ADSSxx is the access controller used by the task node to gain access to the latest dynamic copy of the data.

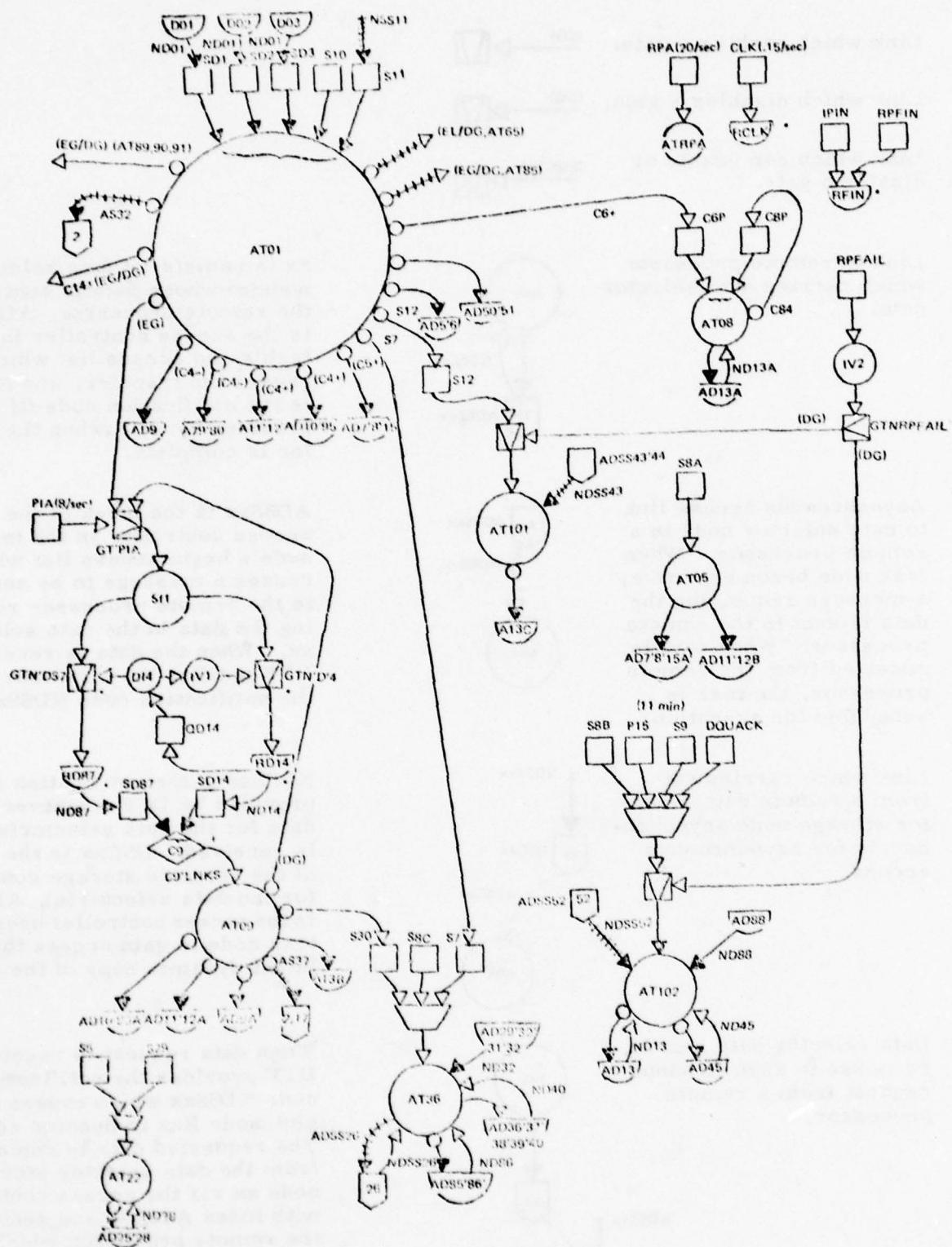
Data selector data sent in response to asynchronous request from a remote processor.

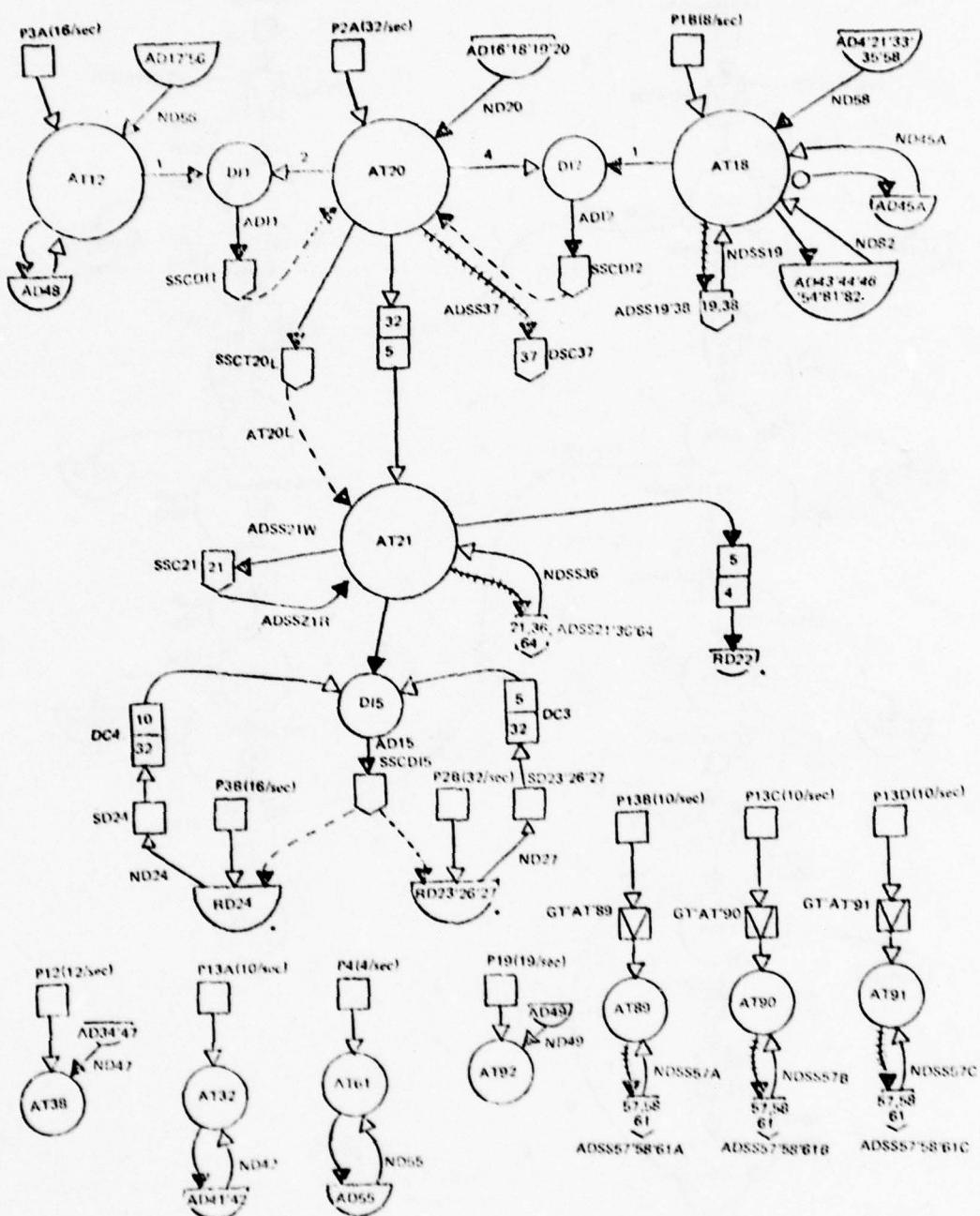


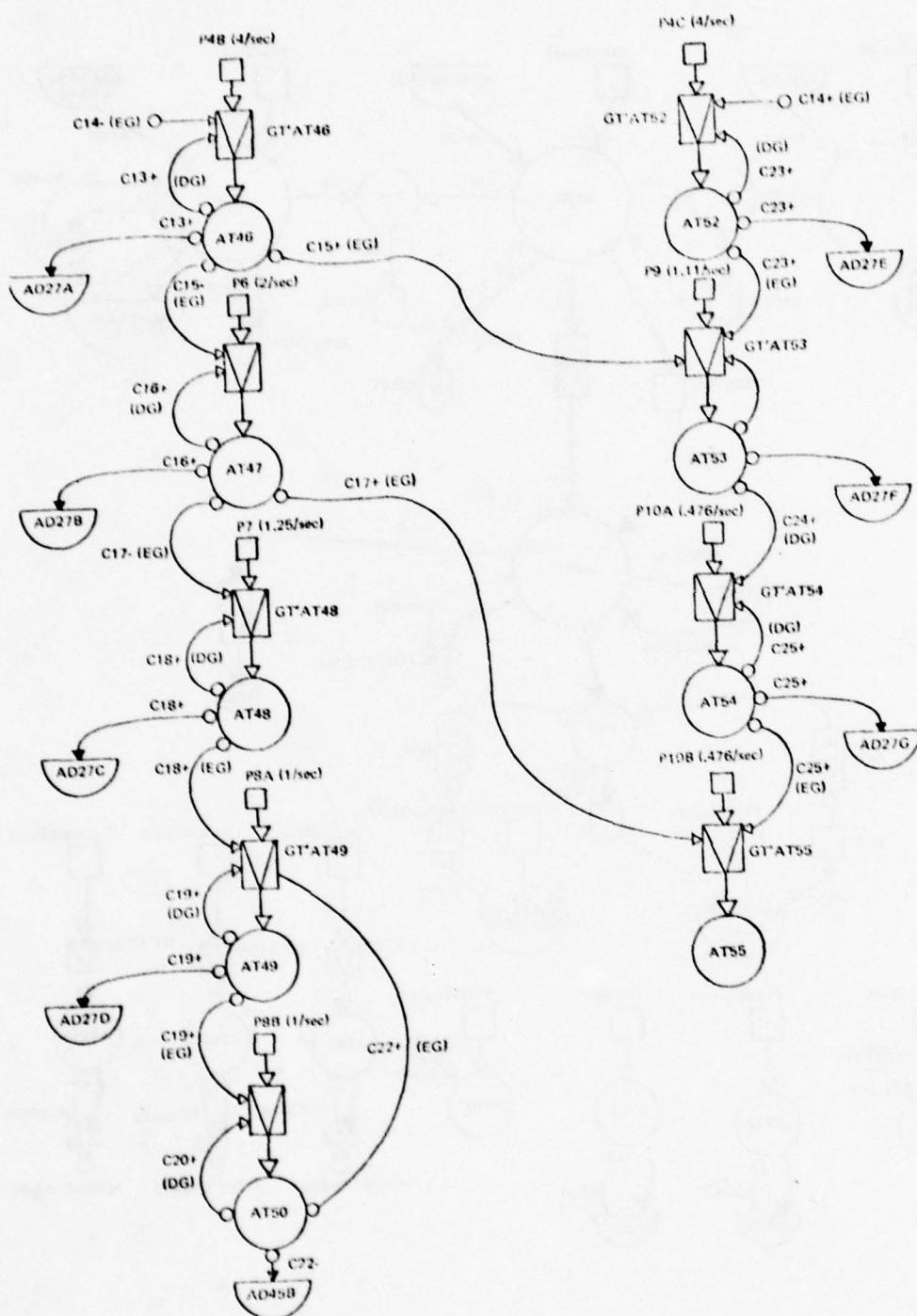
When data request is received, DTF provides the notification code NDSSxx which causes the sink node Rzz to become active. The requested data is copied from the data selector storage node xx via the access controller with index AxxRzz and sent to the remote processor which requested it.

1. 2. 3. 2 Processor 0 DFG Model

PROCESSOR 0 - page 1 of 3

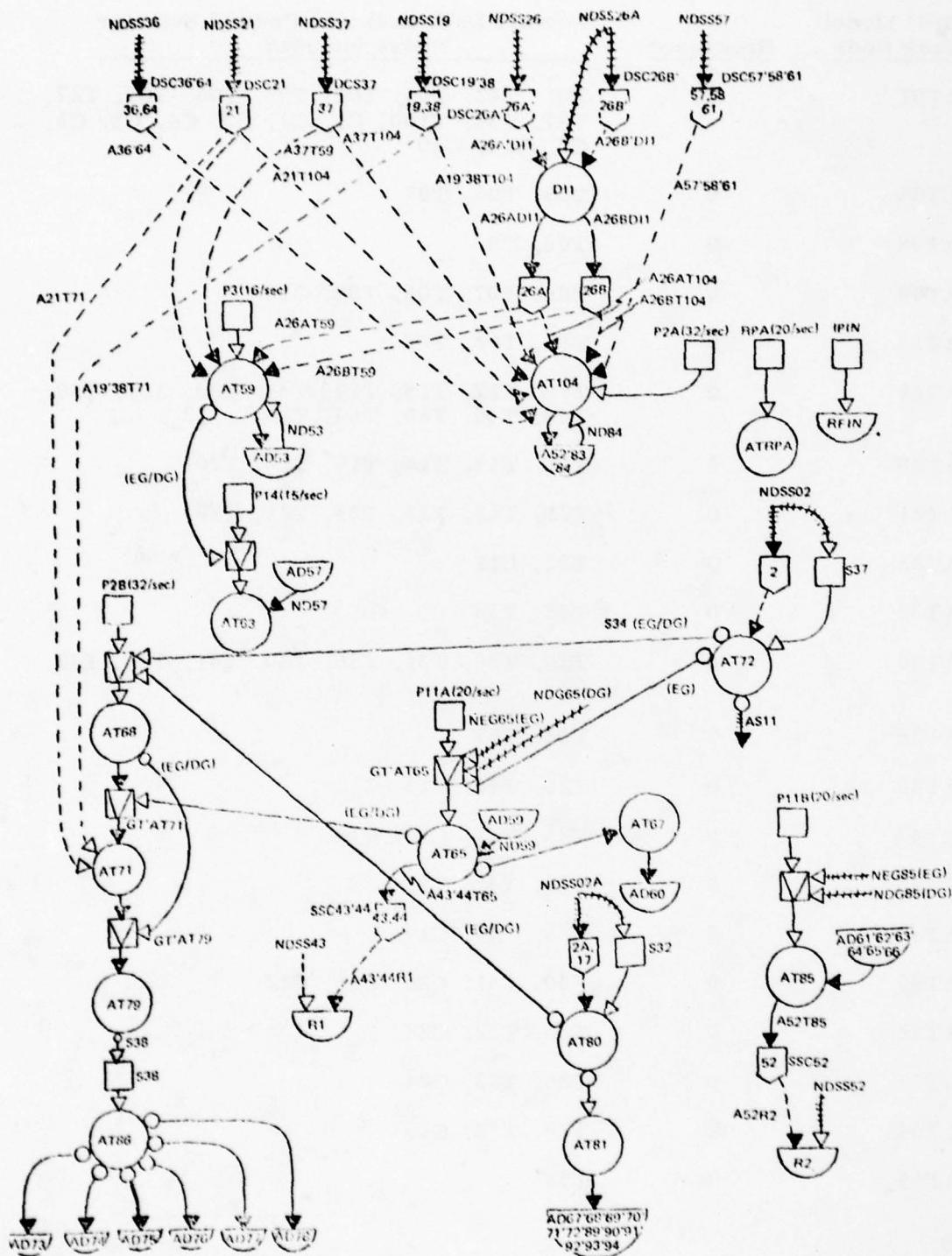






1. 2. 3. 3 Processor 1 DFG Model

PROCESSOR 1 - page 1 of 1



1.2.3.4 Correspondence of DFG Model to Formal DFG

| DFG Model Task Node | Processor | Formal DFG Task and Control Selector Nodes Included |
|--------------------------------|------------------|--|
| AT01 | 0 | T01, T02, T03, T04, T05, T06, T07, T27, T87, T99, T100, C1, C2, C3, C4, C5, C6, C7, C14, C30 |
| AT05 | 0 | T05, T06, T07 |
| AT08 | 0 | T08, C8 |
| AT09 | 0 | T04, T07, T08, T09, C10 |
| AT12 | 0 | T12, T62, T88 |
| AT18 | 0 | T10, T17, T18, T19, T34, T35, T37, T39, T44, T45, T60, T64, T103, C12 |
| AT20 | 0 | T11, T13, T14, T15, T16, T20 |
| AT21 | 0 | T21, T23, T24, T25, T26, T28 |
| AT22 | 0 | T22, C11 |
| AT32 | 0 | T32, T33 |
| AT36 | 0 | T29, T30, T31, T36, T40, T41, T42, T43, C27 |
| AT38 | 0 | T38, T57 |
| AT46 | 0 | T26, T46, C13, C15 |
| AT47 | 0 | T26, T47, C16, C17 |
| AT48 | 0 | T26, T48, C18 |
| AT49 | 0 | T26, T49, C19 |
| AT50 | 0 | T50, T51, C20, C21, C22 |
| AT52 | 0 | T26, T52, C23 |
| AT53 | 0 | T26, T53, C24 |
| AT54 | 0 | T26, T54, C25 |
| AT55 | 0 | T55 |

| <u>DFG Model Task Node</u> | <u>Processor</u> | <u>Formal DFG Task and Control Selector Nodes Included</u> |
|--------------------------------|------------------|--|
| AT59 | 1 | T59 |
| AT61 | 0 | T61 |
| AT63 | 1 | T63 |
| AT65 | 1 | T65, T66 |
| AT67 | 1 | T67 |
| AT68 | 1 | T68 |
| AT71 | 1 | T70, T71, T73 |
| AT72 | 1 | T69, T72, T93, T94, T95, T96, T97, T98 C29 |
| AT79 | 1 | T74, T75, T76, T77, T78, T79 |
| AT80 | 1 | T80 |
| AT81 | 1 | T81, T82, T83, T84 |
| AT85 | 1 | T85 |
| AT86 | 1 | T86 |
| AT89 | 0 | T89 |
| AT90 | 0 | T90 |
| AT91 | 0 | T91 |
| AT92 | 0 | T92 |
| AT101 | 0 | T08, T101 |
| AT102 | 0 | T08, T102 |
| AT104 | 1 | T58, T104 |

Section II

SYSTEM PARAMETERS

The system parameters are those mission parameters which are pertinent to executive performance. The values of these parameters are derived by examination of the constructed DFG model (discussed in Section 1.2) together with the supplementary information provided with the formal mission DFG.

Some data provided with the formal DFG, though necessary in the process of construction of the DFG model, is not directly pertinent to executive performance; for example, the main memory space required by each task. The information which is pertinent are the rates at which clock firings and the asynchronous events which drive the DFG occur. By following the flow of control from clock and external pins on the DFG model the rate at which each node is activated can be estimated. In some cases assumptions about the probable action of control selectors, gates, and tasks which can signal must be made. In general, the worst case conditions (i.e., those which maximize activity) are assumed. In particular, all gates are assumed open unless some mutually exclusive relationship is specified in which case the most severe of the mutually exclusive gating conditions is assumed.

Once this detailed information about node activity has been produced, the rate of each executive activity can be computed. Each executive activity can be timed by straightforward examination of the machine instructions executed. By combining these parameters, the total worst case executive execution time overhead may be computed. In addition, bus loading can be computed based on the activity of nodes which are attached to I/O devices (and remote links) together with the supplementary information to the formal DFG which provides the number of words transferred to or from each device.

Computation of executive space overhead is more easily performed. Program space overhead is simply the total of the space required by each executive procedure. Data space is the total of each data structure size (e.g., table entry size) times the number of instances of that data structure required to implement the tuned DFG model.

2.1 Processor 0 System Parameters

CLOCKS:

| <u>Name</u> | <u>Rate (firings/second)</u> | <u># Pins attached</u> |
|-------------------------|----------------------------------|----------------------------|
| P2 | 32 | 2 |
| RPA | 20 | 1 |
| P19 | 19 | 1 |
| P3 | 16 | 2 |
| P12 | 12 | 1 |
| P13 | 10 | 4 |
| P1 | 8 | 2 |
| P4 | 4 | 3 |
| P6 | 2 | 1 |
| P7 | 1.25 | 1 |
| P9 | 1.11 | 1 |
| P8 | 1.00 | 2 |
| P10 | 0.476 | 2 |
| CLK | 0.150 | 1 |
| <u>P15</u> | <u>0.016</u> | <u>1</u> |
| Totals 15 clocks | 127 firings/sec | 25 clock pins |
| | | 223 pins/sec |

37 simultaneous firings/sec
90 clock interrupts/sec

NODES:

| <u>Type</u> | <u>Quantity</u> | <u>Activations (per second)</u> |
|------------------------|-----------------|-------------------------------------|
| Tasks | | |
| Periodic | 9 | |
| Conditionally periodic | 14 | |
| Aperiodic | 6 | |
| | 29 | 157 |
| Control selector | 1 | 8 |
| Discarder | | |
| 32/5 | 2 | 64 |
| 32/10 | 1 | 16 |
| 5/4 | 1 | 5 |
| Data identity | 5 | 29 |
| Remote link | 7 | 60 |
| Gate | 1 | 0 |
| Simple identity | 1 | 8 |
| Inverter | 2 | 0 |
| Totals | 50 nodes | 347 activations/sec |

LINKS:

| <u>Type</u> | <u>Number Posted</u> | <u>Rate (per second)</u> |
|----------------------|--------------------------|------------------------------|
| Output links | | |
| | 0 | 256 |
| | 1 | 59 |
| | 3 | 32 |
| | | 347 |
| Consume links | | |
| | 0 | 80 |
| Total | | 427 posts/sec |

DATA ACCESS CONTROL:

| <u>Activity</u> | <u>Rate (per second)</u> |
|---------------------------|------------------------------|
| Process begin access list | 246 |
| Process end access list | 246 |
| Process access controller | |
| I/O request | 226 |
| Static copy | 44 |
| Dynamic block allocation | 32 |
| | 302 |
| Word copied by MOV | 461 |

I/O:

| <u>Activity</u> | <u>Rate (per second)</u> |
|--|------------------------------|
| Master to remote switch | 100 |
| Remote to master switch | 100 |
| Master I/O complete notification(s) queued | 90 |
| Remote I/O complete notification(s) queued | 5 |
| I/O complete notification | 216 |
| Data transmitted (command word) | 507 |
| Data word transmitted | 3000 |
| I/O request | |
| Static data | 195 |
| Dynamic data | 32 |
| | 227 |

2.2 Processor 1 System Parameters

CLOCKS:

| <u>Name</u> | <u>Rate (firings/second)</u> | <u># Pins attached</u> |
|-------------|----------------------------------|------------------------------------|
| P2 | 32 | 2 |
| RPA | 20 | 1 |
| P11 | 20 | 2 |
| P3 | 16 | 1 |
| <u>P14</u> | <u>15</u> | <u>1</u> |
| 5 clocks | 103 firings/sec | 7 clock pins |
| | | 155 pins/sec |
| | | <u>36 simultaneous firings/sec</u> |
| | | <u>67 clock interrupts/sec</u> |

NODES:

| <u>Type</u> | <u>Quantity</u> | <u>Activations (per second)</u> |
|------------------------|-----------------|-------------------------------------|
| Tasks | | |
| Periodic | 3 | |
| Conditionally periodic | 6 | |
| Aperiodic | 5 | |
| | 14 | 224 |
| Data identity | 1 | 1 |
| Remote link | <u>3</u> | <u>2</u> |
| Totals | 18 nodes | 227 activations/sec |

LINKS:

| <u>Type</u> | <u>Number posted</u> | <u>Rate (per second)</u> |
|---------------|----------------------|--------------------------|
| Output links | 0 | 163 |
| | 1 | 64 |
| | | 227 |
| Consume links | 0 | <u>227</u> |
| Total | | 454 |

DATA ACCESS CONTROL:

| <u>Activity</u> | <u>Rate (per second)</u> |
|---------------------------|--------------------------|
| Process begin access list | 227 |
| Process end access list | 227 |
| Process access controller | |
| I/O request | 108 |
| Static copy | 40 |
| Begin read dynamic | 352 |
| End read dynamic | 352 |
| | 852 |
| Word copied by MOV | 560 |

I/O:

| <u>Activity</u> | <u>Rate (per second)</u> |
|--|------------------------------|
| Master to remote switch | 100 |
| Remote to master switch | 100 |
| Master I/O complete notification(s) queued | 60 |
| Remote I/O complete notification(s) queued | 50 |
| I/O complete notification | 166 |
| Data transmitted (command word) | 60 |
| Data word transmitted | 561 |
| I/O request (static data) | 110 |

Section III

TUNING THE EXECUTIVE

The OSC executive which results from the construction process described in Section 1 is called the baseline executive for the mission. It is possible that this baseline executive uses so much processor time and space that the mission's resource requirements cannot be met. Hence, it is important that the performance of the baseline executive be calculated and that the executive be tuned if the performance is inadequate.

This section describes in detail the calculation of baseline executive performance for the DAIS mission, the process of tuning the executive, and the performance of the tuned executive.

3.1 Baseline Executive Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 296.096 | 7017 |
| Assembly Language Programs | 74.489 | 1301 |
| Tables | | 3635 |
| Total | 370.585 | 12045 |
| Overhead | 37.1% | 36.8% |
| <u>Processor 1</u> | | |
| J73I Programs | 332.476 | 7017 |
| Assembly Language Programs | 77.882 | 1301 |
| Tables | | 2720 |
| Total | 410.358 | 11130 |
| Overhead | 41.0% | 34% |

This section presents the time and space statistics for the baseline executives. Summaries are presented in Figures 2 and 3. It should be noted that the baseline executive is generalized; that is, it is not dependent on or tailored to the application DFG. The space statistics presented for each cluster consider all programs. Separate figures are presented for the executives containing only the necessary programs.

The executives have not been structured to minimize overhead related to compiler deficiencies since this will be handled in the tuning process. The inability to enable and disable the processing of interrupts as inline functions and the lack of double precision fixed point items requiring assembler procedures for subtracting, adding and comparing these values contributed over 3.5% time overhead to each processor.

| Cluster | Processor 0 | Processor 1 |
|----------|-------------|-------------|
| ITC | 91.436 | 85.591 |
| TIM | 83.011 | 63.532 |
| DAC | 30.123 | 58.165 |
| SCH | 34.635 | 49.415 |
| DTF | 85.775 | 88.803 |
| MSM | 6.829 | 18.983 |
| DSP | 38.776 | 45.869 |
| Total | 370.585 | 410.358 |
| Overhead | 37.1% | 41.0% |

Fig. 2 Baseline Executive Timing Statistics

| Cluster | Total Executive | | Executive with programs used by DFG | |
|----------|-----------------|-------------|--|-------------|
| | Processor 0 | Processor 1 | Processor 0 | Processor 1 |
| ITC | 5093 | 4151 | 4411 | 3129 |
| TIM | 888 | 808 | 810 | 730 |
| DAC | 1387 | 1312 | 1099 | 958 |
| SCH | 332 | 332 | 332 | 332 |
| DTF | 3484 | 3020 | 3484 | 3020 |
| MSM | 426 | 1072 | 426 | 1072 |
| DSP | 435 | 435 | 435 | 435 |
| Total | 12045 | 11130 | 10997 | 9676 |
| Overhead | 36.8% | 34.0% | 33.6% | 29.5% |

Fig. 3 Baseline Executive Space Statistics

3.1.1 ITC Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 88.738 | 3392 |
| Assembly Language Programs | 2.698 | 16 |
| Tables | | 1685 |
| Total | 91.436 | 5093 |
| Overhead | 9.1% | 15.5% |
| <u>Processor 1</u> | | |
| J73I Programs | 83.866 | 3392 |
| Assembly Language Programs | 1.725 | 16 |
| Tables | | 743 |
| Total | 85.591 | 4151 |
| Overhead | 8.6% | 12.7% |

3.1.1.1 Approach

The Intertask Communication cluster is responsible for overall control of DFG interpretation. The primary data structures which control the interpretation are the node table (NDTBL), the pin table (PINTBL), and the link table (LNKTBL). The tasks performed by ITC include:

- Processing of events signaled through pins.
- Posting of enabled and disabled links.
- Processing of active nodes.
- Initiation of I/O activities (via DAC).
- Processing of I/O complete notifications.

3.1.1.2 Definition of ITC Activities

Timing statistics for ITC were derived from the time required to perform each ITC activity together with the rate at which each activity

must be performed. The rates for each activity were obtained directly from the tuned DFG. Since the worst case assumption that all non-mutually exclusive gates were open was made, the timing statistics correspond to peak load conditions.

Pins are signaled primarily through clock firings which occur at the rate of 127 and 103 per second in processors 0 and 1, respectively. Some clocks control more than one pin resulting in 90 and 50 additional pin firings per second. Associated with most of these firings are task starts which occur at the rate of 157 and 224 (worst case) in processors 0 and 1, respectively. In addition, other nodes are activated by pin firings and task completions resulting in a total of 347 and 227 active nodes processed per second (again, worst case).

Notifications of I/O completion occur at the rate of 211 and 105 active notifies, and 5 and 61 passive notifies per second in processors 0 and 1, respectively.

3.1.1.3 ITC Baseline Detailed Timing Statistics

3.1.1.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Secend | Total (milliseconds) |
|----------------------|--|------------------------|---|
| J73I Programs | | | |
| ITCACT | | | |
| Each entry | 18.6 + ENABLE | 90 | 1.674 |
| Each notify | 33.0 | 211 | <u>6.963</u> |
| | | | 8.637 |
| | | | +90 ENABLE |
| ITCPAS | | | |
| Each entry | 18.6 + ENABLE | 5 | 0.093 |
| Each notify | 37.4 | 5 | <u>0.187</u> |
| | | | 0.280 |
| | | | + 5 ENABLE |
| NOTIFY | | | |
| Pin notify | 46.4 | 57 | 2.645 |
| Node notify | 49.2 | 159 | <u>7.823</u> |
| | | | 10.463 |
| ENDTK | 52.8 + ENDTSK + DROP + EACCSS | 133 | 7.022 + 133 ENTSK + 133 DROP + 133 EACCSS |
| ENDTKS | 107.2 + ENDTSK + DRCP + EACCSS + TIME | 24 | 2.573 + 24 ENDTSK + 24 DRCP + 24 EACCSS + 24 TIME |
| ACTIVE | 33.4 | 347 | 11.590 |
| NCSN | 69.4 | 8 | 0.555 |
| NDC | | | |
| 32/5 ratio | | | |
| No output | 51.2 | 54 | 2.765 |
| Output | 59.2 | 10 | <u>0.592</u> |
| 32/10 ratio | | | |
| No output | 51.2 | 11 | 0.563 |
| Output | 59.2 | 5 | <u>0.296</u> |
| 5/4 ratio | | | |
| No output | 51.2 | 1 | <u>0.051</u> |
| Output | 59.2 | 4 | <u>0.237</u> |
| | | | 4.504 |
| NDI | 59.6 + BACCSS + EACCSS | 29 | 1.728 + 29 BACCSS + 29 EACCS |

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-------------------------------------|------------------------|--|
| NRL | 63.8 + BACCSS + EACCSS + SEND | 60 | 3.828 + 60 BACCSS + 60 EACCSS + 60 SEND |
| NSI | 34.6 | 8 | 0.277 |
| NTK | 40.4 + BACCSS + SCHED | 157 | 6.343 + 157 BACCSS + 157 SCHED |
| SIGNAL | | | |
| Each entry | 75.2 | 127 | 9.550 |
| Each additional pin | 48.2 | 91 | <u>4.386</u> |
| | | | 13.936 |
| EPINS | 74.0 | 24 | 1.776 |
| PLINKS | | | |
| Each entry | 26.8 | 427 | 11.444 |
| 1 link posted | 24.4 | 59 | 1.440 |
| 3 links posted | 73.2 | 32 | <u>2.342</u> |
| | | | 15.226 |
| Assembler Programs | | | |
| CNDPRC | 7.6 | 347 | 2.637 |
| EVLP RD | 7.6 | 8 | <u>0.061</u> |
| | | | Total 91.436 |
| | | | + 95 ENABLE |
| | | | + 157 ENDTSK |
| | | | + 157 DROP |
| | | | + 246 EACCSS |
| | | | + 246 BACCSS |
| | | | + 24 TIME |
| | | | + 60 SEND |
| | | | + 157 SCHED |
| | | | Overhead 9.1% |

3. 1. 1. 3. 2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|----------------------|--|------------------------|---|
| J73I Programs | | | |
| ITCACT | | | |
| Each entry | 18.6 + ENABLE | 60 | 1. 116 |
| Each notify | 33.0 | 105 | <u>3. 465</u> |
| | | | 4. 581 |
| | | | + 60 ENABLE |
| ITCPAS | | | |
| Each entry | 18.6 + ENABLE | 50 | 0. 930 |
| Each notify | 37.4 | 61 | <u>2. 281</u> |
| | | | 3. 211 |
| | | | + 50 ENABLE |
| NOTIFY | | | |
| Static pin | 46.4 | 2 | 0. 093 |
| Static node | 49.2 | 105 | 5. 166 |
| Dynamic nil | 80.4 + RETCOR | 61 | 4. 904 |
| Dynamic node | 102.2 + RETCOR | 1 | <u>0. 102</u> |
| | | | 10. 265 |
| | | | + 62 RETCOR |
| ENDTK | 52.8 + ENDTSK + DROP + EACCSS | 121 | 6. 389 + 121 ENDTSK + 121 DRCP + 121 EACCSS |
| ENDTKS | 107.2 + ENDTSK + DROP + EACCSS + TIME | 103 | 11. 042 + 103 ENDTSK + 103 DROP + 103 EACCSS + 103 TIME |
| ACTIVE | 33.4 | 227 | 9. 252 |
| NDI | 59.6 + BACCSS + EACCSS | 1 | 0. 060 + BACCSS + EACCSS |
| NRL | 43.8 + BACCSS + EACCSS | 2 | 0. 088 + 2 BACCSS + 2 EACCSS |
| NTK | 40.4 + BACCSS + SCHED | 224 | 9. 050 + 224 BACCSS + 224 SCHED |

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| SIGNL | | | |
| Each entry | 75.2 | 155 | 11.656 |
| Each additional pin | 48.2 | 50 | <u>2.410</u> |
| | | | 14.066 |
| EPINS | 74.0 | 52 | 3.848 |
| PLINKS | | | |
| Each entry | 26.8 | 390 | 10.452 |
| 1 link posted | 24.4 | 64 | <u>1.562</u> |
| | | | 12.014 |
| Assembler Programs | | | |
| CNDPRC | 7.6 | 227 | <u>1.725</u> |
| | | Total | 85.591 |
| | | | + 110 ENABLE |
| | | | + 62 RETCOR |
| | | | + 224 ENDTSK |
| | | | + 224 DROP |
| | | | + 103 TIME |
| | | | + 227 EACCSS |
| | | | + 227 BACCSS |
| | | Overhead | 8.6% |

3.1.1.4 ITC Detailed Space Statistics

3.1.1.4.1 Processor 0

| | Words |
|---------------------------|------------|
| J73I Programs | |
| ITCINT | 100 |
| EGATES | 28 |
| EGATE | 34 |
| DGATES | 28 |
| DGATE | 32 |
| ITCACT | 48 |
| ITCPAS | 48 |
| NOTIFY | 176 |
| ELGATE | 128 |
| ILNK | 106 |
| ICNDW | 20 |
| DLGATE | 126 |
| RLNK | 128 |
| DCNDW | 22 |
| ENDTK | 48 |
| ENDTKS | 98 |
| ACTIVE | 30 |
| NCSN | 86 |
| NDC | 86 |
| NDI | 70 |
| NIV | 52 |
| NGT | 52 |
| NRL | 76 |
| NSI | 28 |
| NTK | 70 |
| SIGNL | 80 |
| QSIGNAL | 48 |
| DQSIGNAL | 66 |
| EPINS | 78 |
| PLINKS | 58 |
| DLINKS | 36 |
| ENBLGATE | 34 |
| DSBLGATE | 34 |
| SGNLEVENT | 34 |
| SIGAC | 20 |
| Program Data Space | <u>502</u> |
| | 2710 |
| J73I Programs | |
| Not Used by DFG | |
| ENDLG | 24 |
| DNDLG | 28 |
| NCI | 50 |
| NCN | 48 |
| NCS | 110 |
| NDP | 90 |
| NDSS | 96 |

| | |
|-------------|------------|
| NFP | 2 |
| NJN | 66 |
| NGS | 50 |
| DPINS | 84 |
| SIGNLDEVENT | <u>34</u> |
| | <u>682</u> |
| | 3392 |

Assembler Programs

| | |
|----------------|----|
| CNDPRC/EVLP RD | 16 |
|----------------|----|

Data

| | |
|-------------|-------------|
| (see below) | <u>1685</u> |
|-------------|-------------|

| | |
|-------|------|
| TOTAL | 5093 |
|-------|------|

| | |
|----------|-------|
| Overhead | 15.5% |
|----------|-------|

| ITC Data Structure | Size (words) | Occurrences | Total |
|-------------------------------|-----------------|-------------|-----------|
| Nodes (ND) | | | |
| TK nodes | 26 | 20 | 520 |
| TKS nodes | 30 | 11 | 330 |
| nodes | 12 | 5 | 60 |
| RL nodes | 12 | 7 | 84 |
| SI nodes | 8 | 1 | 8 |
| IV nodes | 10 | 2 | 20 |
| GTN nodes | 10 | 3 | 30 |
| DC nodes | 14 | 4 | 56 |
| CSN nodes | 10 | 1 | 10 |
| Padding | 20 | 1 | <u>20</u> |
| | | | 1138 |
| Pins (PIN) | 4 | 43 | 172 |
| Links Vector (LNK) | 1 | 31 | 31 |
| Gates | 4 | 17 | 68 |
| Link Gates (LGT) | 8 | 18 | 144 |
| Notification | | | |
| Controllers (NTF) | 4 | 33 | 132 |
| Misc. Global Variables | 1 | 4 | <u>4</u> |
| | | | Total |
| | | | 1685 |

3. 1. 1. 4. 2 Processor 1

| | Words |
|---------------------------|------------|
| J73I Programs | |
| ITCINT | 100 |
| EGATES | 28 |
| EGATE | 34 |
| DGATES | 28 |
| DGATE | 32 |
| ITCACT | 48 |
| ITCPAS | 48 |
| NOTIFY | 176 |
| ELGATE | 128 |
| ILNK | 106 |
| ICNDW | 20 |
| DLGATE | 126 |
| RLNK | 128 |
| DCNDW | 22 |
| ENDTK | 48 |
| ENDTKS | 98 |
| ACTIVE | 30 |
| NDI | 70 |
| NRL | 76 |
| NTK | 70 |
| SIGNL | 80 |
| QSIGNAL | 48 |
| DQSIGNAL | 66 |
| EPINS | 78 |
| PLINKS | 58 |
| ENBLGATE | 34 |
| DSBLGATE | 34 |
| SIGNLEVENT | 34 |
| SIGAC | 20 |
| Program Data Space | <u>502</u> |
| | 2370 |
| J73I Programs | |
| Not Used by DFG | |
| ENDLG | 24 |
| DNDLG | 28 |
| NCI | 50 |
| NCN | 48 |
| NCS | 110 |
| NCSN | 86 |
| NDC | 86 |
| NDP | 90 |
| NDSS | 96 |
| NFP | 2 |
| NIV | 52 |

| | |
|-------------|-------------|
| NJN | 66 |
| NGS | 50 |
| NGT | 52 |
| NSI | 28 |
| DPINS | 84 |
| DLINKS | 36 |
| SIGNLDEVENT | <u>34</u> |
| | <u>1022</u> |
| | 3392 |

Assembler Programs

| | | |
|---------------------|----|------------|
| CNDPRC/EVLP RD | 16 | 16 |
| Data (see below) | | <u>743</u> |
| Total | | 4151 |
| Overhead | | 12.7% |

| ITC Data Structure | Size | Occurrences | Total |
|-------------------------------------|------|-------------|------------|
| Node Table | | | |
| TK nodes | 26 | 10 | 260 |
| TKS nodes | 30 | 6 | 180 |
| DI nodes | 12 | 1 | 12 |
| RL nodes | 12 | 3 | 36 |
| Padding | 18 | 1 | 18 |
| | | | <u>506</u> |
| Pins | 4 | 15 | 60 |
| Links Vector | 1 | 5 | 5 |
| Gates | 4 | 7 | 28 |
| Link Gates | 8 | 7 | 56 |
| Notification Controllers | 4 | 21 | 84 |
| Misc. Global Variables | 1 | 4 | <u>4</u> |
| | | | <u>743</u> |

3.1.1.5 ITC Sensitivity Analysis

The most important factors affecting ITC overhead are the number of nodes and the rates at which they become active. The rate at which ITC processes active nodes is 347 per second in processor 0 and 227 in processor 1. Handling of these active nodes and posting of their attached links accounts for 55% of the time spent in ITC. In particular, active node dispatching alone (procedure ACTIVE) accounts for approximately 12% of ITC overhead.

Handling of DTF I/O complete notifications at the rates of 216 in processor 0 and 167 in processor 1 accounts for approximately 20% of ITC time.

Signaling of pins proceeds at the rates of 228 and 205 in processors 0 and 1 accounting for approximately 15% of ITC overhead.

3.1.2 TIM Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 45.539 | 589 |
| Assembly Language Programs | 37.472 | 159 |
| Tables | | 140 |
| Total | 83.011 | 888 |
| Overhead | 8.3% | 2.7% |
| <u>Processor 1</u> | | |
| J73I Programs | 28.413 | 589 |
| Assembly Language Programs | 35.119 | 159 |
| Tables | | 60 |
| Total | 63.532 | 808 |
| Overhead | 6.1% | 2.5% |

3.1.2.1 Approach

The Timing cluster has several primary tasks:

- maintain system time
- notify tasks when clock interrupts occur

These tasks are performed by maintaining a list of clocks in each processor. Processor 0 has 15 clocks; processor 1 has 5. Clock A interrupts are fielded by INTCKA, which invokes CLOCKQ to signal the appropriate tasks. It also maintains the clock queue by inserting the clock with the new firing time onto the queue in the appropriate place. The procedure SETCLOCK is called to set timer A to interrupt at the firing time for the first clock on the list. If the time has already passed, SETCLOCK will return a value indicating this.

Clock B is used to synchronize time in each processor. This is done by INTCKB, which is called once every 6.5537 seconds.

3.1.2.2 Definition of TIM Activities

The processor may be in the executive or an application task when an interrupt occurs. There are 127 clock timings per second on the processor 0 DFG, and 103 for processor 1. Of these, at most 90 on processor 0 and 67 on processor 1 are actual interrupts causing INTCKA and CLOCKQ to be entered. The remainder are simultaneous firings for which only one interrupt occurs.

The scheduler must also manipulate times in order to schedule application tasks. In addition, TIME is called by a number of clusters.

3. 1. 2. 3 TIM Baseline Detailed Timing Statistics

3. 1. 2. 3. 1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| CLOCKQ Times entered | 375.7 + ENABLE | 90 | 33.814 |
| Simultaneous firing | 316.9 | 37 | <u>11.725</u> |
| | | | 45.539 |
| | | | + 90 ENABLE |
| Assembler Programs | | | |
| SETCLOCK Timer set (worst case) | 21.8 | 90 | 1.962 |
| TSUM From CLOCKQ | 20.4 | 127 | 2.591 |
| From SCHED | 20.4 | 157 | <u>3.203</u> |
| | | | 5.790 |
| TGTR | | | |
| TRUE from CLOCKQ | 23.8 | 655 | 15.597 |
| FALSE from CLOCKQ | 21.8 | 127 | 2.769 |
| TRUE from SCHED | 23.8 | 236 | 5.617 |
| FALSE from SCHED | 21.8 | 235 | <u>5.123</u> |
| | | | 29.106 |
| TIME | 12.8 | 48 | <u>0.614</u> |
| | | Total | 83.011 |
| | | | + 90 ENABLE |
| | | Overhead | 8.3% |

3. 1. 2. 3. 2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| CLOCKQ | | | |
| Times entered | 296.4 + ENABLE | 67 | 19.859 |
| Simultaneous firing | 237.6 | 36 | <u>8.554</u> |
| | | | 28.413 |
| | | | + 67 ENABLE |
| Assembler Programs | | | |
| SETCLOCK | | | |
| Timer set (worst case) | 21.8 | 67 | 1.461 |
| TSUM | | | |
| From CLOCKQ | 20.4 | 103 | 2.101 |
| From SCHED | 20.4 | 224 | <u>4.570</u> |
| | | | 6.671 |
| TGTR | | | |
| TRUE | | | |
| from CLOCKQ | 23.8 | 285 | 6.783 |
| FALSE | | | |
| from CLOCKQ | 21.8 | 103 | 2.245 |
| TRUE | | | |
| from SCHED | 23.8 | 336 | 7.997 |
| FALSE | | | |
| from SCHED | 21.8 | 336 | <u>7.325</u> |
| | | | 24.350 |
| TIME | 12.8 | 206 | <u>2.637</u> |
| | | | |
| | | Total | 63.532 |
| | | | + 67 ENABLE |
| | | Overhead | 6.4% |

3.1.2.4 TIM Baseline Detailed Space Statistics

3.1.2.4.1 Processor 0

| | Words |
|-------------------------|-------------|
| J73I Programs | |
| TIMINT | 20 |
| STARTCLOCK | 112 |
| CLOCKQ | 154 |
| SETALARM | 102 |
| CLRALAR - (not used) | 78 |
| Program Data Space | <u>123</u> |
| | 589 |
| Assembler Programs | |
| Interrupt handlers | 40 |
| SETCLOCK | 24 |
| TIME | 16 |
| TSUM/TDIF/TGTR | 64 |
| Program Data Space | <u>15</u> |
| | 159 |
| Data | |
| Clock Table | 136 |
| Miscellaneous items | <u>4</u> |
| Total | <u>140</u> |
| Overhead | 888 2.7% |

3.1.2.4.2 Processor 1

| | |
|---------------------|-------------|
| J73I Programs | |
| Same as Processor 0 | 589 |
| Assembler Programs | |
| Same as Processor 0 | 159 |
| Data | |
| Clock Table | 56 |
| Miscellaneous items | <u>4</u> |
| Total | <u>60</u> |
| Overhead | 808 2.5% |

3.1.2.5 Sensitivity Analysis

The prime determinant of overhead is the structure of the clock queue. Each attempt to insert a clock on the list requires 57.0 microseconds for each clock on the list with an earlier firing time. Processors 0 and 1 average 5.16 and 2.77 clocks, respectively, for each insert. This is 3.7% overhead on processor 0 to insert clocks 127 times each second, and 1.6% overhead for processor 1.

Two other factors affecting overhead are beating clocks and missed firing times. When two or more clocks fire at the same time, only one clock interrupt is taken. When a firing time is missed, the interrupt is processed even though the physical interrupt did not occur. Both of these actions take less time than actual interrupts.

3.1.3 DAC Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 28.479 | 1044 |
| Assembly Language Programs | 1.644 | 16 |
| Tables | | 327 |
| Total | 30.123 | 1387 |
| Overhead | 3.0% | 4.2% |
| <u>Processor 1</u> | | |
| J73I Programs | 56.389 | 1044 |
| Assembly Language Programs | 1.776 | 16 |
| Tables | | 252 |
| Total | 58.165 | 1312 |
| Overhead | 5.8% | 4.0% |

3.1.3.1 Approach

The Data Access Control cluster manages access to global data (data accessed by more than one task), and initiation of I/O activities (via DTF). DAC interfaces with ITC via the interface procedures BACCSS and EACCSS. These procedures process lists of access controllers which contain information about the type of access required. According to the access type, control is dispatched within DAC to process each requested access operation.

3.1.3.2 Definition of DAC Activities

Processing of nodes (primarily task nodes) by ITC results in 246 and 227 calls to BACCSS and EACCSS per second in processors 0 and 1,

respectively. Each call involves the processing of an access list resulting in a number of accesses of each possible type occurring each second. The rates at which each access type occurs were collected from the rate of each node activation together with the accesses required by that node as specified on the tuned DFG.

3.1.3.3 DAC Baseline Detailed Timing Statistics

3.1.3.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------------------|
| J73I Programs | | | |
| BACCSS | | | |
| Each entry | 20.4 | 246 | 5.018 |
| Each AC processed | 24.8 | 130 | <u>3.224</u> |
| | | | 8.242 |
| EACCSS | | | |
| Each entry | 20.4 | 246 | 5.018 |
| Each AC processed | 24.8 | 172 | <u>4.226</u> |
| | | | 9.284 |
| BWD | 42.8 + GETCOR | 32 | 1.370 + 32 GETCOR |
| ESWS | 48.6 | 34 | 1.652 |
| BSRS | 68.4 | 10 | 0.684 |
| CSTN | 38.0 + SEND | 8 | 0.304 + 8 SEND |
| CST | 33.4 + SEND | 25 | 0.835 + 25 SEND |
| USTN | 32.0 + SEND | 151 | 4.832 + 151 SEND |
| UST | 25.8 + SEND | 10 | 0.258 + 10 SEND |
| UDT | 31.8 + SEND | 32 | 1.018 + 32 SEND |
| Assembler Programs | | | |
| CYPD | | | |
| Each entry | 16.4 | 44 | 0.722 |
| Words copied | 2.0 | 461 | <u>0.922</u> |
| | | | <u>1.644</u> |
| | | Total | 30.123 + 32 GETCOR + 226 SEND |
| | | Overhead | 3.0% |

3.1.3.3.2 Processor 1

| Processor | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|---|--------------------------------|---------------------------------|
| J73I Programs | | | |
| BACCSS | | | |
| Each entry | 20.4 | 227 | 4.631 |
| Each AC processed | 24.8 | 407 | <u>10.094</u> |
| | | | <u>14.725</u> |
| EACCSS | | | |
| Each entry | 20.4 | 227 | 4.631 |
| Each AC processed | 24.8 | 445 | <u>11.036</u> |
| | | | <u>15.667</u> |
| BARD | 31.8 | 352 | 11.194 |
| ERD | 26.8 | 352 | 9.434 |
| ESWS | 48.6 | 40 | 1.944 |
| USTN | 32.0 + SEND | 103 | 3.296 + 103 SEND |
| UST | 25.8 + SEND | 5 | 0.129 + 5 SEND |
| Assembler Programs | | | |
| CPYD | | | |
| Each entry | 16.4 | 40 | 0.656 |
| Words copied | 2.0 | 560 | <u>1.120</u> |
| | | | <u>1.776</u> |
| | | Total | 58.165 + 108 SEND |
| | | Overhead | 5.8% |

3.1.3.4 DAC Baseline Detailed Space Statistics

3.1.3.4.1 Processor 0

Words

J73I Programs

| | |
|--------|-----|
| DACINT | 150 |
| BACCSS | 70 |
| EACCSS | 100 |
| BARS | 36 |
| BWD | 42 |
| ESWS | 46 |
| CSTN | 34 |
| CST | 30 |
| USTN | 28 |
| UST | 22 |
| UDT | 34 |

| | |
|---------------------------|------------|
| Program Data Space | 158 |
| | 756 |

J73I Programs

Not Used by DFG

| | |
|-------|------------|
| BARD | 34 |
| BSRD | 22 |
| BSRS | 20 |
| EA WD | 54 |
| EAWS | 38 |
| ERD | 40 |
| ESWD | 80 |
| | 288 |

Assembler Programs

| | | |
|-------------|-----------|-----------|
| CPYD | 16 | 16 |
|-------------|-----------|-----------|

Data

| | |
|--------------------|-------------------|
| (see below) | <u>327</u> |
|--------------------|-------------------|

| | |
|--------------|-------------|
| Total | 1387 |
|--------------|-------------|

| | |
|-----------------|-------------|
| Overhead | 4.2% |
|-----------------|-------------|

| DAC Data Structure | Size (words) | Occurrences | Total |
|--|---------------------|--------------------|--------------|
| Access Controllers (A) | 4 | 62 | 248 |
| Dynamic Storage Controllers (DSC) | 4 | 1 | 4 |
| Static Storage Controllers (SSC) | 4 | 5 | 20 |
| Extra Static Blocks (SSB) | | | |
| for extra copies of certain data blocks | 4 | 3 | 12 |
| | 17 | 1 | 17 |
| | 26 | 1 | 26 |
| | | | 55 |
| | | | 327 |

3.1.3.4.2 Processor 1

| Words | | | |
|--|---------|-------------|-----------|
| J73I Programs | | | |
| | | | |
| DACINT | 150 | | |
| BACCSS | 76 | | |
| EACCSS | 100 | | |
| BARD | 34 | | |
| BARS | 36 | | |
| ERD | 40 | | |
| ESWS | 46 | | |
| USTN | 28 | | |
| UST | 22 | | |
| Program Data Space | 158 | | 690 |
| | | | |
| J73I Programs | | | |
| Not Used by DFG | | | |
| BSRD | 22 | | |
| ESRS | 20 | | |
| BWD | 42 | | |
| EA WD | 54 | | |
| EA WS | 38 | | |
| ESWD | 80 | | |
| CSTN | 34 | | |
| CST | 30 | | |
| UDT | 34 | | 354 |
| | | | |
| Assembler Programs | | | |
| | | | |
| CPYD | | 16 | |
| | | | |
| Data | | | |
| (see below) | | | |
| Total | | <u>252</u> | |
| Overhead | | 1312 | |
| | | 4. 0% | |
| | | | |
| DAC Data Structure | | | |
| | | | |
| Access Controllers (AC) | Size | Occurrences | Total |
| Dynamic Storage Controllers (DSC) | 4 | 36 | 144 |
| Static Storage Controllers (SSC) | 4 | 10 | 40 |
| Extra Static Blocks (SSB) for extra copies of certain data blocks | 8 20 | 2 2 | 16 40 |
| | | | <u>56</u> |
| | | | 252 |

3.1.3.5 DAC Sensitivity Analysis

The important factors contributing to DAC overhead are the number of access lists processed per second and the number of access controllers in these access lists. Management of the processing of these lists (not including the processing of each individual access controller) accounts for approximately 55% of DAC overhead while individual access controller processing accounts for the remaining 45%.

In processor 0, 24% of DAC time is spent processing I/O request access controllers while in processor 1 only 6% is spent here. In both processors the rates at which static data is copied is fairly low; substantial increases in these rates would be required if more frequent task pre-emption were allowed. For example, if 400 static blocks averaging 10 words had to be copied per second, DAC overhead in processor 0 would nearly double. In processor 1 a substantial amount of read access to dynamic data is handled (352 accesses per second) accounting for 35% of DAC overhead.

3.1.4 SCH Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 34.635 | 332 |
| Assembly Language Programs | | |
| Tables | | |
| Total | 34.635 | 332 |
| Overhead | 3.5% | 1.0% |
| <u>Processor 1</u> | | |
| J73I Programs | 49.415 | 332 |
| Assembly Language Programs | | |
| Tables | | |
| Total | 49.415 | 332 |
| Overhead | 4.9% | 1.0% |

3.1.4.1 Approach

The Scheduling cluster orders the execution of application tasks. It interfaces with ITC and DAC via procedure SCHED which is called to schedule a task for execution. SCH maintains a queue structure which holds all scheduled tasks not yet executed and all tasks which have been partially executed and then preempted. Tasks in the scheduling queue are ordered by deadline and by preemption rules.

When a task completes, DROP is called by ITC to remove the task from the scheduling queue. The dispatcher (DSP) examines the top of the scheduler queue and either restarts the active task or starts a new task. If a new task is started, AUTASK is called to manipulate the queue so that the task is on the active task stack.

3.1.4.2 Definition of Activities

The important parameter affecting SCH overhead is the number of task starts per second (157 in processor 0, and 224 in processor 1). Each task execution implies one SCHED call, one AUTASK call, and one DROP call. Calls to SPRTIM (two per task start) are not included in the baseline timing statistics but are discussed separately in Section 3.1.4.5.

At each SCHED call the queue is searched until the first pre-emptable task is found (usually the idle task). The new task is inserted into the queue of tasks scheduled to preempt that task according to its deadline.

3.1.4.3 SCH Baseline Detailed Timing Statistics

3.1.4.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|------------------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| SCCHED | | | |
| Each entry | 106.8 + TSUM | 157 | 16.768 |
| Each task not preemptable | + TGTR | | |
| Each task | 28.4 | 157 | 4.459 |
| more urgent | 24.2 + TGTR | 157x2 | <u>7.599</u> |
| | | | 28.826 |
| | | | + 157 TSUM |
| | | | + 471 TGTR |
| AUTASK | 12.4 | 157 | 1.947 |
| DROP | 24.6 | 157 | <u>3.862</u> |
| | | Total | 34.635 |
| | | | + 157 TSUM |
| | | | + 471 TGTR |
| | | Overhead | 3.5% |

3.1.4.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|----------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| SCCHED | | | |
| Each entry | 106.8 + TSUM | 224 | 23.923 |
| Each task | + TGTR | | |
| not preemptable | 28.4 | 224 | 6.362 |
| Each task | | | |
| more urgent | 24.2 + TGTR | 224x2 | <u>10.842</u> |
| | | | 41.127 |
| | | | + 224 TSUM |
| | | | + 672 TGTR |
| AUTASK | 12.4 | 224 | 2.778 |
| DROP | 24.6 | 224 | <u>5.510</u> |
| | | Total | 49.415 |
| | | | + 224 TSUM |
| | | | + 672 TGTR |
| | | Overhead | 4.9% |

3.1.4.4 SCH Baseline Detailed Space Statistics

3.1.4.4.1 Processor 0

| | Words |
|---------------------------|-----------|
| J73I Programs | |
| SCHINT | 12 |
| SCHEDE | 120 |
| AUTASK | 12 |
| DROP | 28 |
| SPRTIME | 88 |
| Program Data Space | 72 |
| | 332 |
| Data | |
| None | <hr/> |
| Total | 332 |
| Overhead | 1.0% |

3.1.4.4.2 Processor 1

| | Words |
|---------------------|-------|
| J73I Code | |
| Same as Processor 0 | 332 |
| Data | |
| None | <hr/> |
| Total | 332 |
| Overhead | 1.0% |

3.1.4.5 SCH Sensitivity Analysis

Scheduling is the most frequently occurring activity (157 per second in processor 0, 224 per second in processor 1) which requires a search. However, the size of the queue at any given time is not particularly large since it only includes tasks ready to run which have not been run. Hence much of the scheduler overhead (approximately 50%) is due to loop setups in SCHED and actual insertion of tasks into the queue. The total time required by SCH is proportional to the number of task starts/sec.

Part of the work performed in scheduling is the numerous calls to TSUM and TGTR to perform double precision functions. The overhead for these calls was included in the TIM cluster baseline timing statistics. Had it been included with SCH, the SCH overhead would be approximately 40% higher than it was in both processors.

Overhead involved in computing spare time for all tasks in the schedule queue was not included because it is essentially a testing tool. The overhead was, however, computed separately and the timing statistics are presented below. Overhead for calls to TIME, TSUM, TGTR and TDIF are included.

Spare Time Computation Timing Statistics

Processor 0

SPRTIME

| | | | |
|--------------------|-------|-------------|----------------|
| Each entry | 53.0 | 157 x 2 | 16.642 |
| Each task in queue | 122.4 | 157 x 2 x 3 | <u>115.300</u> |
| | | Total | 131.943 |
| | | Overhead | 13.2% |

Processor 1

SPRTIME

| | | | |
|--------------------|-------|-------------|----------------|
| Each entry | 53.0 | 224 x 2 | 23.744 |
| Each task in queue | 122.4 | 224 x 2 x 3 | <u>164.506</u> |
| | | Total | 188.250 |
| | | Overhead | 18.8% |

3.1.5 DTF Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 57.515 | 1192 |
| Assembly Language Programs | 18.260 | 946 |
| Tables | | 1346 |
| Total | 85.775 | 3484 |
| Overhead | 8.6% | 10.6% |
| <u>Processor 1</u> | | |
| J73I Programs | 70.543 | 1198 |
| Assembly Language Programs | 18.260 | 946 |
| Tables | | 882 |
| Total | 88.803 | 3020 |
| Overhead | 8.9% | 9.2% |

3.1.5.1 Approach

Processor 0 is specified as the head processor. This requires it to perform certain tasks not done by processor 1. These include:

- Synchronizes system time in the other processor.
- Coordinates system initialization by taking master control of the bus, sending an initialization signal, and passing bus control to the nonhead processor.

Tasks that are performed by both processors include:

- Error recovery and retry associated with bus commands.
- Passing control of the bus from processor to processor according to a deadline priority scheme.
- Dynamic construction of command word lists from a fixed memory list as events occur.

Two of these tasks are the prime contributors to time overhead:

- Passing bus control uses 4.8% of processor 0, 7.0% of processor 1.
- Construction of command word lists uses 4.0% of processor 0, 1.9% of processor 1.

3.1.5.2 Definition of DTF Activities

The procedure BCI2 is responsible for controlling the bus. There is a tradeoff between the overhead associated with the number of times it passes bus control and the ability to meet response requirements if the bus is held for too long. The timing statistics are based on an average time for holding the bus of five milliseconds, thereby taking control and relinquishing it 100 times each second. The average time holding the bus is controlled by two factors; the length of the command word list and the amount of bus fill time introduced. Bus fill time is the time the processor with bus master control uses the bus to receive dummy data from the other processor. The bus fill time can be set within the BCI2 program.

The system response requirements are such that an average bus hold time in excess of ten milliseconds would be adequate, thus cutting the control passing overhead by more than one half. However, it is felt that a five millisecond average would be more representative for most systems.

3.1.5.3 DTF Baseline Detailed Timing Statistics

3.1.5.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|---|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| DTFACT | | | |
| ITCACT to be queued | 45.8 + QUEUE | 90 | 4.122 |
| No ITCACT notification requirement | 38.8 | 10 | .388 |
| Dynamic storage data transmitted | 21.6 + RETCOR | 32 | <u>.691</u> |
| | | | 5.201 |
| | | | + 32 RETCOR calls |
| | | | + 90 QUEUE calls |
| DTFPAS | | | |
| Each entry | 158.0 | 100 | 15.800 |
| ITCPAS to be queued | 1.4 + QUEUE | 5 | .007 |
| Each block of data rec'd | 27.4 | 5 | <u>.137</u> |
| | | | 15.944 |
| | | | + 5 QUEUE calls |
| DTFKEY | 63.4 | 100 | 6.340 |
| SENDYN | 66.6 | 32 | 2.130 |
| SEND | | | |
| Allocated entry | 76.2 | 195 | 14.860 |
| Allocated additional command word/entry | 55.8 | 262 | 14.620 |
| Allocated notify | 15.8 | 195 | 3.080 |
| Unallocated entry | 104.6 | 40 | 4.180 |
| Unallocated additional command word/entry | 84.2 | 10 | .840 |
| Unallocated notify | 15.8 | 20 | <u>.320</u> |
| | | | 37.900 |
| Assembler Programs | | | |
| BCI2 | | | |
| Take control of bus | 114.6 | 100 | 11.460 |
| Relinquish control of bus | 68.0 | 100 | <u>6.800</u> |
| | | | 18.260 |
| | | Total | 85.775 |
| | | Overhead | 8.6% |

3.1.5.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|---|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| DTFACT | | | |
| ITCACT to be queued | 45.8 + QUEUE | 60 | 3.984 |
| No ITCACT notification req. | 38.8 | 40 | <u>1.552</u> |
| | | | 5.536 |
| | | | + 60 QUEUE calls |
| DTFPAS | | | |
| Each entry | 364.0 | 100 | 36.400 |
| ITCPAS to be queued | 1.4 + QUEUE | 50 | .070 |
| Each block of statically allocated data received | 27.4 | 5 | .137 |
| Each block of dynamically allocated data received | 47.4 + GETCOR | 60 | <u>2.844</u> |
| | | | 39.451 |
| | | | + 60 GETCOR calls |
| | | | + 50 QUEUE calls |
| DTFKEY | 63.4 | 100 | 6.340 |
| SEND | | | |
| Each entry | 76.2 | 110 | 8.382 |
| Each additional command word/entry | 55.8 | 165 | 9.207 |
| Each notify | 15.8 | 100 | <u>1.627</u> |
| | | | 19.216 |
| Assembler Programs | | | |
| BCI2 | | | |
| Take control of bus | 114.6 | 100 | 11.460 |
| Relinquish control of bus | 68.0 | 100 | <u>6.800</u> |
| | | | 18.260 |
| | | | Total 88.803 |
| | | | Overhead 8.9% |

3.1.5.4 DTF Baseline Detailed Space Statistics

3.1.5.4.1 Processor 0

Words

J73I Programs

| | |
|--------------------|-----|
| RDYBUS | 24 |
| DTFINT | 190 |
| DOWN | 52 |
| FLIP | 168 |
| DTFPAS | 208 |
| DTFACT | 102 |
| DTFKEY | 62 |
| SENDYN | 62 |
| SEND | 208 |
| Program Data Space | 116 |

1192

Assembler Programs

BCIU interrupt handlers to transfer control,
synchronize time, support monitoring of
other processors, do all bus message retries
and determine failures

| | |
|-----------------------------------|-----|
| BCIU initialization | 88 |
| BCIU interrupt and error handling | 846 |
| setting new priority | 12 |

946

Data

| | |
|---|-----|
| DEVTBL - Device list table | 36 |
| CWTBL - Command word table | 636 |
| CWQCWP - Queued CW indexes | 41 |
| MNOTFY - Master/ITC notification buffer | 110 |
| RNOTFY - Remote/ITC notification buffer | 24 |
| R CODE - Remote/ITC notification codes | 5 |
| FLPDEV - Failed device CW queue | 40 |
| Miscellaneous Items J73I | 21 |
| Subaddress Pointer Words | 118 |
| Storage for Executive 'Signals' | 6 |
| Scratch Storage | 32 |
| Priorities [left as 4 Processor Case] | 5 |
| Miscellaneous Items Assembler | 32 |
| Command Word Storage | 240 |

1346

| | |
|----------|-------|
| Total | 3484 |
| Overhead | 10.6% |

3.1.5.4.2 Processor 1

J73I Programs

| | |
|---------------------|------|
| Same as Processor 0 | 1192 |
|---------------------|------|

Assembler Programs

| | |
|---------------------|-----|
| Same as Processor 0 | 946 |
|---------------------|-----|

Data

| | |
|---|------------|
| DEVTBL - Device list table | 40 |
| CWTBL - Command word table | 246 |
| CWQCWP - Queued CW indexes (not used) | 1 |
| MNOTFY - Master/ITC notification buffer | 30 |
| RNOTFY - Remote/ITC notification buffer | 70 |
| RCODE - Remote/ITC notification codes | 15 |
| FLPDEV - Failed Device CW Queue | 30 |
| Miscellaneous Items J73I | 21 |
| Subaddress Pointer Words | 110 |
| Storage for Executive 'Signals' | 12 |
| Scratch Storage | 32 |
| Priorities [left as 4 Processor case] | 5 |
| Miscellaneous Items Assembler | 30 |
| Command Word Storage | 240 |
| | <u>882</u> |
| Total | 3020 |
| Overhead | 9.2% |

3.1.5.5 Sensitivity Analysis

There are two primary factors affecting the overhead of DTF. The first is the tradeoff between the overhead associated with each bus control transfer and meeting response requirements. The amount of overhead increases linearly with each transfer of control; however, if the number of transfers is reduced, the average length of time the bus is controlled between transfer of control increases and it becomes more difficult to meet response requirements.

The second factor is the amount of data that must be transferred between processors. The effect on the overhead can be seen by examining the executives. Each time control is received, DTFPAS is entered. In processor 0, where up to five blocks of data can be received, each entry takes 158.0 microseconds to process, contributing a total of 1.6% to overhead. However, processor 1 can receive up to fifteen blocks of data. This requires 364.0 microseconds to process for a total of 3.6% overhead. This is an increase of 2%, directly attributable to the additional ten blocks of data that can be transferred.

3.1.6 MSM Baseline Statistics

| Processor | Time/second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 6.829 | 258 |
| Assembly Language Programs | | |
| Tables | | 68 |
| Total | 6.829 | 426 |
| Overhead | .7% | 1.2% |
| <u>Processor 1</u> | | |
| J73I Programs | 18.989 | 358 |
| Assembly Language Programs | | |
| Tables | | 714 |
| Total | 18.989 | 1072 |
| Overhead | 1.9% | 3.2% |

3.1.6.1 Approach

Each executive maintains a list of free blocks. Each allocated block will be 34 words long and use space from the first block on the list. When a block of storage is returned, it is placed in the list in ascending order by address. If a block is contiguous at either end, it will be compacted with the contiguous block to keep one larger block on the list.

3.1.6.2 Definition of MSM Activities

When storage is allocated by a call to GETCOR, it comes from the first block of storage that is large enough. The block may be the exact size required, or it may be larger leaving a smaller block of the remaining space on the list.

When a block of storage is returned, it may be contiguous to other blocks on the free list. In addition, it is placed in the appropriate place on the list.

In processor 0, there are at most two GETCOR calls in a row followed by two RETCOR calls. In processor 1, there are fourteen GETCOR calls. This is followed by at most seven RETCOR calls before any storage is returned.

3.1.6.3 MSM Baseline Detailed Timing Statistics

3.1.6.3.1 Processor 0

| Procedure J73I Programs | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|--|-----------------------------------|------------------------|-------------------------|
| GETCOR | 101.0 | 32 | 3.232 |
| RETCOR | | | |
| First on list, contiguous at end | 109.4 | 24 | 2.626 |
| First on list, not contiguous at end | 105.0 | 4 | .420 |
| Second on list, contiguous both sides | 137.8 | 4 | .551 |
| | | | <u>3.597</u> |
| | | Total | 6.829 |
| | | Overhead | .7% |

3.1.6.3.2 Processor 1

| Procedure J73I Programs | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|----------------------------|-----------------------------------|------------------------|-------------------------|
| GETCOR | | | |
| Exact fit | 151.6 | 31 | 4.700 |
| Extra space | 160.8 | 31 | 4.985 |
| | | | <u>9.685</u> |
| RETCOR | | | |
| Not contiguous | 137.8 | 15 | 2.067 |
| Contiguous both ends | 154.2 | 15 | 2.313 |
| Contiguous front | 149.8 | 16 | 2.397 |
| Contiguous end | 142.2 | 16 | 2.275 |
| | | | <u>9.298</u> |
| | | Total | 18.983 |
| | | Overhead | 1.9% |

3.1.6.4 MSM Baseline Detailed Space Statistics

3.1.6.4.1 Processor 0

| | Words |
|----------------------|------------|
| J73I Programs | |
| GETCOR | 142 |
| RETCOR | 166 |
| MSMINT | 18 |
| Program Data Space | 32 |
| | 358 |
| Data | |
| Two blocks | <u>68</u> |
| Total | 426 |
| Overhead | 1.2% |

3.1.6.4.2 Processor 1

| | Words |
|----------------------|------------|
| J73I Programs | |
| GETCOR | 142 |
| RETCOR | 166 |
| MSMINT | 18 |
| Program Data Space | 32 |
| | 358 |
| Data | |
| Twenty-one blocks | <u>714</u> |
| Total | 1072 |
| Overhead | 3.2% |

3.1.6.5 Sensitivity Analysis

The primary factor affecting both space and time is the sequencing of calls to GETCOR and RETCOR. If the calls are interspersed and random, then a free list is created. As the free list gets longer, the time to process and the number of blocks of storage required increases linearly.

3.1.7 DSP Baseline Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | 24.361 | 110 |
| Assembly Language Programs | 14.415 | 164 |
| Tables | | 69 |
| Total | 38.776 | 343 |
| Overhead | 3.9% | 1.0% |
| <u>Processor 1</u> | | |
| J73I Programs | 24.867 | 110 |
| Assembly Language Programs | 21.002 | 164 |
| Tables | | 69 |
| Total | 45.869 | 343 |
| Overhead | 4.6% | 1.0% |

3.1.7.1 Approach

The Dispatching cluster controls the assignment of the processor to executive and application tasks. Tasks to be run in executive mode are queued (by interrupt handlers) through calls (some inline) to QUEUE. Whenever an executive task completes it returns through the DSP program DQUEUE which dispatches the next executive task in that queue until it is empty.

When all executive tasks are complete, the dispatcher either restarts the application task which was interrupted or starts a new more urgent application task.

3.1.7.2 Definition of DSP Activities

The dispatcher tends to be the most active cluster in the executive. Entries via DQUEUE occur at each interrupt (200 bus and 90 clock in processor 0 and 200 bus, 67 clock in processor 1) and at each task completion (157 in processor 0 and 224 in processor 1). Hence, assuming worst case, tasks are restarted 290 and 267 times a second, and started 157 and 224 times a second in processor 0 and 1 respectively. Restarting interrupted or preempted tasks involves manipulation of the stack and restoring of all registers and the machine state. Starting of new tasks requires a call to the schedule function AUTASK and allocation of a stack frame.

DSP also contains assembly programs ENABLE and DISABL which are called by other clusters.

3.1.7.3 DSP Baseline Detailed Timing Statistics

3.1.7.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| QUEUE | 20.6 | 95 | 1. 957 |
| DQUEUE | | | |
| Each entry | | | |
| Task ends | | 157 | |
| DTF interrupt returns | | 200 | |
| TIM interrupt returns | | <u>90</u> | |
| | 16.2 | 447 | 7. 241 |
| Each item queued | | | |
| DTF queueing | | 95 | |
| Interrupts in executive | | <u>87</u> (30% of 290) | |
| | 24.8 | 182 | <u>4. 514</u> |
| | | | 11. 755 |
| DSPTSK | | | |
| Interrupt returns | 26.0 | 290 | 7. 540 |
| Task starts | 19.8 + AUTASK | 157 | <u>3. 109</u> |
| | | | 10. 649 |
| | | | + 157 AUTASK |
| Assembler Programs | | | |
| CEXCPR | 6.0 | 182 | 1. 092 |
| RSTART | 24.6 | 290 | 7. 134 |
| START | 8.8 | 157 | 1. 382 |
| ENDTSK | 7.8 | 157 | 1. 225 |
| ENABLE | 4.4 | 185 | 0. 814 |
| DISABL | 4.4 | 629 | <u>2. 768</u> |
| | | Total | 38. 776 |
| | | Overhead | 3. 9% |

3.1.7.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| J73I Programs | | | |
| QUEUE | 20.6 | 110 | 2.266 |
| DQUEUE | | | |
| Each entry | | | |
| Task ends | | 224 | |
| DTF interrupt returns | | 200 | |
| TIM interrupt returns | | 67 | |
| 16.2 | | 491 | 7.954 |
| Each item queued | | | |
| DTF queueing | | 110 | |
| Interrups in executive | | 80 (30% of 267) | |
| 24.8 | | 190 | 4.712 |
| | | | 12.666 |
| DSPTSK | | | |
| Interrupt returns | 26.0 | 267 | 5.500 |
| Task starts | 19.8 + AUTASK | 224 | 4.435 |
| | | | 9.935 |
| | | | + 224 AUTASK |
| Assembler Programs | | | |
| CEXCPR | 6.0 | 190 | 6.840 |
| RSTART | 24.6 | 267 | 6.568 |
| START | 8.8 | 224 | 1.971 |
| ENDTSK | 7.8 | 224 | 1.747 |
| ENABLE | 4.4 | 200 | 0.880 |
| DISABL | 4.4 | 681 | 2.996 |
| | | Total | 45.869 |
| | | Overhead | 4.6% |

3.1.7.4 DSP Baseline Detailed Space Statistics

3.1.7.4.1 Processor 0

| | Words |
|--|-----------|
| J73I Programs | |
| DSPINT | 78 |
| QUEUE | 16 |
| DQUEUE | 36 |
| DSPTSK | 26 |
| ENDTSK | 10 |
| Program Data Space | 36 |
| | 202 |
| Assembler Programs | |
| ENABLE/DISABL | 8 |
| CEXCPR | 6 |
| START/RSTART | 18 |
| Machine initialization/ Fault handling | 132 |
| | 164 |
| Data | |
| EXCQ - executive task queue | 6 |
| Initialization handler transfer addresses | 32 |
| Machine initialization/ fault handling | 31 |
| | <u>69</u> |
| Total | 435 |
| Overhead | 1.3% |

3.1.7.4.2 Processor 1

| | Words |
|---------------------|-----------|
| J73I Programs | |
| Same as Processor 0 | 202 |
| Assembler Programs | |
| Same as Processor 0 | 164 |
| Data | |
| Same as Processor 0 | <u>69</u> |
| Total | 435 |
| Overhead | 1.3% |

3.1.7.5 DSP Sensitivity Analysis

Approximately 40% of the time spent in the dispatcher is spent in queueing and dequeuing executive tasks. This operation is triggered at each task completion and each interrupt. These occur at the combined rate of 447 per second in processor 0 and 491 per second in processor 1 (worst case). Approximately 30% is consumed in restarting interrupted application tasks, and 20% in starting and finishing tasks. The remaining 10% is consumed in performing the interrupt enable and disable functions for various J73I programs in the executive.

3.1.8 SSM Baseline Statistics

Secondary Storage Management is not implemented.

3.1.9 MPL Baseline Statistics

Multiprocessor Locking is not implemented.

3.1.10 Bus Traffic

There are two events which control the amount of activity on the bus. The first is passing control of the bus. This requires sending two data words, one command word and one status word. In addition, there is a delay of 57.2 microseconds each time control is passed. Control is passed 100 times each second.

The second event causing bus traffic is the transmission of data. Each transmission causes a command word, status word and the data word to be sent. There is a transmission for each command word allocated on calls to SEND. The number of data words sent is derived from the data associated with each command word.

Each data word requires 20 microseconds to send.

The statistics for each processor are summarized in the following sections. The transmissions include both device and interprocessor data.

AD-A055 902 SOFTECH INC WALTHAM MASS F/G 9/2
OPERATIONAL SOFTWARE CONCEPT OSC EXECUTIVE EVALUATION/REFINEMEN--ETC(U)
AUG 77 M G WILLOUGHBY, C K HITCHON F33615-76-C-1192

UNCLASSIFIED

1025-3

AFAL-TR-77-87

NL

2 of 2
AD
A055 902



END

DATE

FILMED

8-78

DDC

3.1.10.1 Processor 0

| | Occurrences /Second | Words Transmitted /Second |
|--------------------|------------------------|------------------------------|
| Passing control | 100 | 400 |
| Data transmissions | 507 | 1014 |
| <u>Data words</u> | | <u>3000</u> |
| | | 4414 words |
| | | 2.0 microseconds/word |
| | | 88.28 milliseconds |
| Idle time | | <u>5.72</u> milliseconds |
| Total | | 94.00 |
| Overhead | | 9.4% |

3.1.10.2 Processor 1

| | Occurrences /Second | Words Transmitted /Second |
|-------------------|------------------------|------------------------------|
| Passing control | 100 | 400 |
| Data transmission | 275 | 550 |
| <u>Data words</u> | | <u>6500</u> |
| | | 7450 words |
| | | 2.0 microseconds/word |
| Idle Time | | 149.0 milliseconds |
| | | <u>5.72</u> milliseconds |
| Total | | 154.72 |
| Overhead | | 15.5% |

3.1.10.3 Interprocessor Bus Traffic

3.1.10.3.1 Processor 0 to Processor 1

| | Occurrences /Second | Words Transmitted /Second |
|--|------------------------|------------------------------|
| Data transmissions | 60 | 120 |
| Data words | | 561 |
| Negligible asynchronous transmissions | | |
| | | <u>681</u> |

3.1.10.3.2 Processor 1 to Processor 0

Negligible - all asynchronous transmissions

| | |
|-------------------------------------|---|
| Total of all interprocessor traffic | 681 words 20 microseconds/word 13.62 milliseconds |
| Overhead | 1.4% |

3.2 Tuning for HOL Inefficiencies

3.2.1 Overview

This section provides examples of efficiency gain by recoding J73I executive procedures into DAIS assembly language. The approach taken is to recode procedures such that they could be inserted into the executive as a replacement for the J73I procedures. Further, recoding is done with the same limitations that apply to the compilers; primarily, procedure linkage conventions must be adhered to, procedure calls must be made, and state variables may be modified across procedure calls.

The intention of this section is to obtain an understanding of the extra OSC executive overhead associated with inefficiencies in code generated by the compiler.

3.2.2 Examples

Examples have been selected from three executive function clusters: CLOCKQ from the TIM cluster, SEND from the DTF cluster and QUEUE and DQUEUE from the DSP cluster. These examples were selected as representative of all features of the code that comprises the executive. Timing statistics are based on the system performance parameters of processor 0.

3.2.2.1 CLOCKQ

The hand coded version of CLOCKQ is shown in Appendix A, Figure A.1. Overhead comparisons between the baseline and handcoded versions are as follows:

| | Time/Second (milliseconds) | Space (words) |
|---|-------------------------------|------------------|
| Baseline Version | 45.539 | 182 |
| Handcoded Version | 20.085 | 102 |
| Percentage Reduction over Baseline Version | 55.9% | 44.0% |

3.2.2.2 SEND

The handcoded version of SEND is shown in Appendix A, Figure A.2.

Overhead comparisons between the baseline and handcoded versions are as follows:

| | Time/Second (milliseconds) | Space (words) |
|---|-------------------------------|------------------|
| Baseline Version | 37.900 | 240 |
| Handcoded Version | 21.123 | 149 |
| Percentage Reduction over Baseline Version | 44.3% | 37.9% |

3.2.2.3 QUEUE and DQUEUE

The handcoded versions of QUEUE and DQUEUE are shown in Appendix A, Figure A.3.

Overhead comparisons between the baseline and handcoded versions are as follows:

| | Time/Second (milliseconds) | Space (words) |
|---|-------------------------------|------------------|
| Baseline version | 13.712 | 61 |
| Handcoded version | 8.504 | 39 |
| Percentage reduction over Baseline Version | 38.0% | 36.1% |

3.2.3 Summary

The effect on the OSC executive of eliminating compiler code inefficiency can be estimated as a 48.2% reduction in time and a 40.0% reduction in space. This estimate is derived from the composite efficiency gain shown in the previous three examples.

3.3 Tuning by Reducing Generality

3.3.1 Overview

This section provides examples of efficiency gain resulting from removing executive generality not required for a specific DFG.

3.3.2 Examples

The examples used and the efficiency comparisons made are based on the results of Section 3.2, Tuning for HOL Inefficiencies.

3.3.2.1 CLOCKQ

The excess generality of CLOCKQ with respect to the processor 0 DFG requirements exists only in the ability to handle alarm clocks. The recoded version of CLOCKQ without alarm clock capability is shown in Appendix A, Figure A.4. Comparisons between the handcoded version of Section 3.2 and the recoded version of this section are as follows:

| | Time/Second (milliseconds) | Space (words) |
|--|-------------------------------|------------------|
| Handcoded version | 20.085 | 102 |
| No excess generality version | 19.374 | 98 |
| Percentage Reduction over handcoded version | 3.5% | 3.9% |

3.3.2.2 SEND

The processor 0 DFG does not require SEND to support unallocated master receive and remote receive subaddresses. The recoded version of SEND with the above capability removed is shown in Appendix A, Figure A.5. Comparisons between the handcoded version and the version of this section

are as follows:

| | Time/Seconds (milliseconds) | Space (words) |
|--|--------------------------------|------------------|
| Handcoded version | 21. 123 | 149 |
| No excess generality version | 19. 915 | 88 |
| Percentage Reduction over handcoded version | 5. 7% | 40. 9% |

3. 3. 2. 3 QUEUE and DQUEUE

QUEUE and DQUEUE contain no excess generality.

3. 3. 3 Summary

The removal of excess generality from the executive is estimated to yield an efficiency improvement of 3. 9% in time and 22. 4% in space. However, the space reduction is closely coupled with the complexity of the DFG. Extremely complex DFGs may require all the features of the executive and allow for no generality reduction while simple DFGs could allow on the order of a 50% or more space reduction.

The structure of the executive is such that additional capability can be added relatively easily with an impact on space rather than time. The results of this section tend to validate that statement.

3.4 Tuning for HOL Language Deficiencies

3.4.1 Overview

This section provides examples of efficiency gain that would result from additional language features in the J73I compiler. The primary language features desired for the OSC executive are a built-in function (BIF) capability and a more powerful inline capability.

The built-in function capability would allow certain machine instructions to be accessed by the HOL directly rather than through procedure calls to assembly language programs. The efficiency gain is largely derived by having the necessary instructions generated in-line, thereby circumventing the procedure linkage overhead and the required register flushing. The built-in functions most desired for the OSC executive are interrupt enabling and disabling, jump (branch to an address), and double precision fixed point arithmetic (although a double precision fixed point language feature would be more desirable).

An inline feature using an INLINE directive would be much more desirable from the inline point of view rather than the somewhat obscure and inflexible DEFINE.

3.4.2 Examples

The examples used in this section are derived from the same procedures used in the two previous sections.

3.4.2.1 CLOCKQ

CLOCKQ makes extensive use of double precision fixed point arithmetic. The large efficiency gain observed is due primarily to performing this arithmetic inline. The recoded version of this section is shown in Appendix A, Figure A.6.

Comparisons between the CLOCKQ version of Section 3.3 and the recoded version of this section are as follows:

| | Time/Second (milliseconds) | Space (words) |
|--|-------------------------------|------------------|
| No excess generality version | 19.374 | 98 |
| HOL deficiencies removed version | 10.223 | 62 |
| Percentage reduction over no excess generality version | 47.2% | 36.7% |

3.4.2.2 SEND

The recoding of SEND to eliminate HOL deficiencies shows limited improvement. The improvement comes mainly from the ability to manage registers more effectively due to the absence of procedure calls. The recoded version of SEND is shown in Appendix A, Figure A.7.

Comparisons between the SEND version of the previous section and the version of this section is as follows:

| | Time/Second (milliseconds) | Space (words) |
|--|-------------------------------|------------------|
| No excess generality version | 19.915 | 88 |
| HOL deficiencies removed version | 16.957 | 83 |
| Percentage reduction over no excess generality version | 14.9% | 5.7% |

3.4.2.3 QUEUE and DQUEUE

The recoding of QUEUE and DQUEUE showed no change in QUEUE and a marked improvement in DQUEUE due to including DSPTSK inline and making a direct jump to the queued procedure rather than through a procedure call to CEXCPR.

Comparisons between the handcoded versions of QUEUE and DQUEUE from Section 3.2 (no change was observed by eliminating excess generality) and the version of this section are as follows:

| | Time/Second (milliseconds) | Space (words) |
|--|-------------------------------|------------------|
| Handcoded version | 8.504 | 39 |
| HOL deficiencies removed version | 4.386 | 27 |
| Percentage reduction over handcoded version | 48.6% | 30.8% |

3.4.3 Summary

The removal of the language deficiencies described in this section from the J73I compiler would yield an estimated efficiency improvement of 34.0% in time and 23.6% in space for the OSC executive.

3.5 Final Tuning

3.5.1 Overview

The tuning steps described in sections 3.2 through 3.4 yielded an efficiency gain of 67.5% in time and 64.4% in space for the selected examples. The composite results of the stepwise tuning for CLOCKQ, SEND, QUEUE, and DQUEUE are shown below.

| | Time/Second (milliseconds) | Space (words) |
|---|-------------------------------|------------------|
| Baseline | 97.151 | 483 |
| Tuned by removing the inefficiencies and deficiencies and excess generality | 31.566 | 172 |
| Percentage reduction over baseline version | 67.5% | 64.4% |

The following performance can be expected by applying the results of the three-step tuning process to the executives of processor 0 and processor 1. The space statistics reflect a reduction in program space only. Data structures were not tuned.

| | Processor 0 Executive Overhead | | Processor 1 Executive Overhead | |
|----------|--------------------------------|-------|--------------------------------|-------|
| | Time | Space | Time | Space |
| Baseline | 37.1% | 33.7% | 41.0% | 29.2% |
| Tuned* | 12.1% | 20.1% | 13.3% | 17.3% |

* Tuning limited to removing HOL inefficiencies and deficiencies and excess generality.

The final tuning phase involves global modifications to the executives that result from the three-step tuning process.

These global modifications are described in the following subsection.

3.5.2 Final Tuning Description

The final tuning process involves examining the executive as a whole, and studying the flow of data and control between its parts in an effort to reduce or eliminate the overhead involved in communication among these various parts. In performing this process, the functional boundaries delineated by function clusters and individual procedures may be blurred and in some cases lost altogether. The result of final tuning is therefore not only a smaller, faster executive but also an executive whose parts are intricately interwoven. Such an executive is likely to be difficult to understand, modify, and debug.

In the case of tuning the OSC executive for the DAIS mission, the final tuning process was not necessary to meet mission requirements but was performed in order to demonstrate the method and measure of its efficacy. The estimated reduction in executive time overhead resulting from the final tuning process was about 9.0% of the tuned overhead or only 2.7% of the baseline version overhead. That is, 3.9% of the total overhead reduction is attributable to final tuning and 96.1% to all other tuning methods. In view of this and the drawbacks of final tuning mentioned above, the process should be applied only when mission requirements cannot otherwise be met.

The types of optimization applied in the final tuning process include:

- Global allocation of registers throughout the executive. Global allocation of registers reduces and in some cases eliminates the need for movement of operands back and forth between the register file and main memory, even over procedure calls. State-of-the-art compilers are capable of this type of optimization but generally limit the allocation to local variables within a single procedure.

Examples of this global allocation are the dedication of a register to the active node stack (ANODE) throughout ITC processing and the dedication of a register to the executive control queue (EXCQ) throughout the executive.

- Inline expansion of calls between function clusters. Here the meaning of the term inline expansion is extended to mean the absorption of the actions of part or all of one function cluster into another function cluster. For example, in the final tuned version, CLOCKQ (in the TIM function cluster) performs the functions of SIGNL (in the ITC function cluster) by stacking the nodes activated by each clock firing and then transferring control to ITC to process the active node stack when all fired clocks have been processed. Note this is not the same as a direct inline expansion of the call to SIGNL in CLOCKQ. A direct expansion would result in a call to ITC to process active nodes for every clock firing thus incurring more TIM/ITC linkage overhead.
- Elimination of procedure call/return linkages where possible. In many cases execution of executive procedures follow one another in a predictable sequence. In such cases it is possible to reduce overhead by substituting direct transfers for the usual procedure call/return mechanism. For example, in the baseline executive, ENDTK always calls ACTIVE and then DQUEUE which either transfers control to another executive task or to the dispatcher but never returns. There is no need for ACTIVE to return to ENDTK or for DQUEUE to return to ACTIVE. In the final tuned version, ENDTK transfers directly to active node processing which in turn transfers directly to DQUEUE when the active node stack is empty. DQUEUE itself transfers directly to the next executive procedure on the executive control queue (EXCQ) or, if the queue is empty, then to DSPTSK which dispatches an application task. Naturally some flexibility may be lost in that such procedures are inseparably tied to a particular flow of control and therefore may not be called individually.

3.6 Final Tuned Executive

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 1678 |
| Assembly Language Programs | 110.730 | 2007 |
| Tables | | 3558 |
| Total | 110.730 | 7243 |
| Overhead | 11.1% | 22.1% |
| <u>Processor 1</u> | | |
| J73I Programs | | 1708 |
| Assembly Language Programs | 113.220 | 2095 |
| Tables | | 2482 |
| Total | 113.220 | 6285 |
| Overhead | 11.3% | 19.2% |

The tuned executives were created by using each of the tuning methods previously described. Summary statistics are provided in Figures 4 and 5. The following sections will provide detailed statistics for each cluster.

| Cluster | Processor 0 | Processor 1 |
|----------|-------------|-------------|
| ITC | 23.628 | 24.741 |
| TIM | 16.610 | 11.150 |
| DAC | 9.001 | 17.903 |
| SCH | 8.330 | 12.141 |
| DTF | 39.354 | 32.306 |
| MSM | 0.000 | 0.000 |
| DSP | 13.811 | 14.982 |
| Total | 110.734 | 113.223 |
| Overhead | 11.1% | 11.3% |

Fig. 4 Final Tuned Executive Timing Statistics

| Cluster | Processor 0 | Processor 1 |
|----------|-------------|-------------|
| ITC | 3257 | 2315 |
| TIM | 368 | 288 |
| DAC | 699 | 624 |
| SCH | 60 | 60 |
| DTF | 2589 | 2082 |
| MSM | 83 | 729 |
| DSP | 187 | 187 |
| Total | 7243 | 6285 |
| Overhead | 22.1% | 19.2% |

Fig. 5 Final Tuned Executive Space Statistics

3.6.1 ITC Final Tuned Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 970 |
| Assembly Language Programs | 23.628 | 602 |
| Tables | | 1685 |
| Total | 23.628 | 3257 |
| Overhead | 2.4% | 9.9% |
| <u>Processor 1</u> | | |
| J73I Programs | | 970 |
| Assembly Language Programs | 24.741 | 602 |
| Tables | | 743 |
| Total | 24.741 | 2315 |
| Overhead | 2.5% | 7.1% |

3.6.1.1 ITC Tuning Approach

The reductions in ITC functionality described below account for much of the space reduction and a significant improvement in speed. All frequently called procedures were written in assembly language. In general, calls to small procedures in other function clusters were expanded into inline assembly code. In particular, this was done for calls to ENDTSK, DROP, RETCOR, and TIME. Internal function cluster procedures ACTIVE, EPINS, EGATES, and DGATES were also expanded inline. The interface procedure SIGNL was eliminated altogether, thus requiring the TIM cluster to perform this function inline. Procedure linkages were simplified and in some cases eliminated altogether. For example, each active node processing procedure simply sends control to the active node procedure for the next node on the active stack. A

dummy node always at the bottom of the stack automatically transfers control to DQUEUE when all active nodes have been processed. Registers were allocated on an executive wide basis, and saving and reloading around procedure calls was eliminated.

Data structures owned by ITC were not modified (e.g., node table, pin table, links vector, gate table). However, some reduction in table space (especially the node table) could have been achieved by removing data items required only by unsupported functions (e.g., ND'ETIME which contains a task node's maximum execution time).

3.6.1.2 Functional Differences with Baseline

- Only node types TK, TKS, CSN, GT, SI, DI, DC, IV, RL are required.
- Of the node types implemented, only GT checks for disabled inputs.
- Only the '+' link of the CSN node is implemented.
- No consume links are posted.
- Only pin and links block gates are implemented.
- A links block gate may modify only the LINKS'BLK links block.
- A maximum of four output links are allowed for each node.
- RL node does not invoke EACCSS.

3.6.1.3 ITC Final Tuned Detailed Timing Statistics

3.6.1.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------------------|
| Assembler Programs | | | |
| ITCACT | | | |
| Each entry | 18.2 | 90 | 1.638 |
| Each notify | 11.4 | 211 | <u>2.405</u> |
| | | | 4.043 |
| ITCPAS | | | |
| Each entry | 18.2 | 5 | 0.091 |
| Each notify | 11.4 | 5 | <u>0.057</u> |
| | | | 0.148 |
| NOTIFY | | | |
| Pin notify | 31.4 | 57 | 1.790 |
| Node notify | 20.4 | 159 | <u>3.244</u> |
| | | | 5.034 |
| ENDTK | 37.4 + EACCSS | 133 | 4.974 + 133 EACCSS |
| ENDTKS | 74.6 + EACCSS | 24 | 1.790 + 24 EACCSS |
| NCSN | 16.4 | 8 | 0.131 |
| NDC | | | |
| 32/5 | | 32x2 | |
| No output | 18.2 | 54 | 0.983 |
| Output | 26.2 | 10 | 0.262 |
| 32/10 | | 16 | |
| No output | 18.2 | 11 | 0.200 |
| Output | 26.2 | 5 | 0.131 |
| 5/4 | | 5 | |
| No output | 18.2 | 1 | 0.018 |
| Output | 26.2 | 4 | <u>0.105</u> |
| | | | 1.699 |
| NDI | 24.4 + BACCSS + EACCSS | 29 | 0.708 + 29 BACCSS + 29 EACCSS |
| NRL | 13.8 + BACCSS + SEND | 60 | 0.828 + 60 BACCSS + 60 SEND |
| NSI | 16.4 | 8 | 0.131 |

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|------------------|-----------------------------------|------------------------|--|
| NTK | 14.6 + BACCSS + SCHED | 157 | 2.292 + 157 BACCSS + 157 SCHED |
| PLINKS | | | |
| Each 0 link post | 2.0 | 98 | 0.196 |
| Each 1 link post | 12.4 | 59 | 0.732 |
| Each 3 link post | 28.8 | 32 | <u>0.922</u> |
| | | | 1.850 |
| | | Total | 23.628 + 246 BACCSS + 186 EACCSS |
| | | Overhead | 2.4% |

3.6.1.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Time (milliseconds) |
|---------------------------|-----------------------------------|------------------------|------------------------|
| Assembler Programs | | | |
| ITCACT | | | |
| Each entry | 18.2 | 60 | 1.092 |
| Each notify | 11.4 | 105 | <u>1.197</u> |
| | | | 2.289 |
| ITCPAS | | | |
| Each entry | 18.2 | 50 | 0.910 |
| Each notify | 11.4 | 66 | <u>0.752</u> |
| | | | 1.662 |
| NOTIFY | | | |
| Static pin notify | 31.4 | 2 | 0.063 |
| Static gate notify | 10.0 | 2 | 0.020 |
| Static node notify | 20.4 | 105 | 2.142 |
| Dynamic nil notify | 30.8 | 61 | 1.879 |
| Dynamic node notify | 41.2 | 1 | <u>0.041</u> |
| | | | 4.145 |
| ENDTK | 37.4 | 121 | 4.525 |
| ENDTKS | 74.6 | 103 | 7.684 |
| NTK | 14.6 | 224 | 3.270 |
| NDI | 24.4 | 1 | 0.024 |
| NKL | 13.8 | 2 | <u>0.028</u> |
| PLK0 | 2.0 | 160 | 0.320 |
| PLK1 | 12.4 | 64 | <u>0.794</u> |
| | | Total | 24.741 |
| | | Overhead | 2.5% |

3.6.1.4 ITC Final Tuned Detailed Space Requirements

3.6.1.4.1 Processor 0

| | Words |
|---------------------------|-------------------|
| J73I Programs | |
| ITCINT | 100 |
| ELGATE | 128 |
| ILINK | 106 |
| ENDLG | 24 |
| ICNDW | 20 |
| DLGATE | 126 |
| RLNK | 128 |
| DNDLG | 28 |
| DCNDW | 22 |
| ENBLGATE | 34 |
| DSBLGATE | 34 |
| SIGNLEVENT | 34 |
| SIGNLDEVENT | 34 |
| SIGAC | 20 |
| Program Data Space | <u>132</u> |
| | 970 |

Assembler Programs

| | |
|----------|----|
| ITCACT | 30 |
| ITCPAS | 30 |
| NOTIFY | 84 |
| NDI | 24 |
| NIV | 20 |
| NGT | 22 |
| NRL | 18 |
| NCSN | 24 |
| NDC | 26 |
| EGATE | 14 |
| DGATE | 14 |
| DQSIGNAL | 30 |

Assembler Programs (Continued)

| | |
|--------------------|-----------|
| DLINKS | 26 |
| PLINKS | 46 |
| ENDTK | 42 |
| ENDTKS | 104 |
| NSI | 16 |
| NTK | 16 |
| Program Data Space | <u>16</u> |
| | 602 |

Data

| | |
|------------------|-------------|
| Same as baseline | <u>1685</u> |
| Total | 3257 |
| Overhead | 9.9% |

3.6.1.4.2 Processor 1

Words

| | |
|---------------------------|-----|
| J73I Programs | |
| Same as processor 0 | 970 |
| Assembler Programs | |
| Same as processor 0 | 602 |

Data

| | |
|------------------|------------|
| Same as baseline | <u>743</u> |
| Total | 2315 |
| Overhead | 7.1% |

3.6.1.5 ITC Sensitivity Analysis

The time overhead in the final tuned ITC is about 28% of the baseline version's overhead in both processors. A slightly higher percentage (60%-70%) of ITC time is spent in processing active nodes than in the baseline. This is due to absorption of segments of other functions into inline code, in particular the posting of clock pins. Most of the active node processing is of task nodes; hence a doubling of the number of tasks or their rates of execution would result in approximately a 50% increase in ITC overhead, or 1.2% of total processor overhead. The remaining 30% to 40% is consumed in processing I/O complete notifications.

Program space is dramatically reduced by the elimination of unnecessary generality and the compaction resulting from hand coding.

3.6.2 TIM Final Tuned Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 158 |
| Assembly Language Programs | 16.610 | 70 |
| Tables | | 140 |
| Total | 16.610 | 368 |
| Overhead | 1.7% | 1.1% |
| <u>Processor 1</u> | | |
| J73I Programs | | 158 |
| Assembly Language Programs | 11.150 | 70 |
| Tables | | 60 |
| Total | 11.150 | 288 |
| Overhead | 1.1% | 0.9% |

3.6.2.1 TIM Timing Approach

The functions TIME, TGTR, TSUM, TDIF were expanded as inline assembly code wherever called and therefore are not implemented as procedures in this cluster. SETCLOCK and SETALRM were merged into CLOCKQ which was recoded in assembly language. The interface between CLOCKQ and ITC was modified so that the functions performed by SIGNL in ITC were done inline in CLOCKQ. Instead of calling ACTIVE for each enabled pin, CLOCKQ stacks the activated nodes and exits by transferring control to the active node procedure for the first node on the stack.

A more efficient method of handling the timer interrupt (INTCKA) was devised. This method of handling interrupts is discussed in section 3.6.7.1.

The TIM owned data structures were not modified.

3.6.2.2 Functional Differences with Baseline

- One shot alarms are not implemented.

3.6.2.3 TIM Final Tuned Detailed Timing Statistics

3.6.2.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|-----------------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| INTCKA | | | |
| From application task | 24.0 | 81 | 1. 944 |
| From executive task | 15.0 | 9 | <u>0. 135</u> |
| | | | 2. 079 |
| CLOCKQ | | | |
| Entries | 13.2 | 90 | 1. 188 |
| Clock firings | 42.8 | 127 | 5. 436 |
| Multiple pin clocks | 18.6 | 96 | 1. 786 |
| Pass first clock on list | 16.2 | 127 | 2. 057 |
| Pass additional clocks on list | 8.0 | 508 | <u>4. 064</u> |
| | | | <u>14. 531</u> |
| | | Total | 16. 610 |
| | | Overhead | 1. 7% |

3.6.2.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|-----------------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| INTCKA | | | |
| From application task | 24.0 | 60 | 1. 440 |
| From executive task | 15.0 | 9 | <u>0. 135</u> |
| | | | 1. 575 |
| CLOCKQ | | | |
| Entries | 13.2 | 67 | 0. 884 |
| Clock Firings | 42.8 | 103 | 4. 408 |
| Multiple Pin Clocks | 18.6 | 52 | 0. 967 |
| Pass first clock on list | 10.2 | 103 | 1. 668 |
| Pass additional clocks on list | 8.0 | 206 | <u>1. 648</u> |
| | | | <u>9. 575</u> |
| | | Total | 11. 150 |
| | | Overhead | 1. 1% |

3.6.2.4 TIM Final Tuned Detailed Space Statistics

3.6.2.4.1 Processor 0

| | | Words |
|---------------------------|------------|-------|
| J73I Programs | | |
| TIMINT | 20 | |
| STARTCLOCK | 112 | |
| Program Data Space | <u>26</u> | |
| | 158 | |
| Assembler Programs | | |
| CLOCKQ | <u>70</u> | |
| | 70 | |
| Data | | |
| Same as baseline | <u>140</u> | |
| Total | 368 | |
| Overhead | 1.1% | |

3.6.2.4.2 Processor 1

| | | Words |
|---------------------------|-----------|-------|
| J73I Programs | | |
| Same as processor 0 | 158 | |
| Assembler Programs | | |
| Same as processor 0 | 70 | |
| Data | | |
| Same as baseline | <u>60</u> | |
| Total | 288 | |
| Overhead | 0.9% | |

3.6.2.5 TIM Sensitivity Analysis

The time overhead in the final tuned version is about 20% of the untuned version's overhead. Searching the clock list to insert a clock which has just fired accounts for about 30% of the processing. Additional independent clocks result in more clocks firing per second and more clocks to search through on the queue after each firing. Hence, TIM overhead is a function of the product of firings/second and the number of independent clocks.

3.6.3 DAC Final Tuned Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 172 |
| Assembly Language Programs | 9.001 | 200 |
| Tables | | 327 |
| Total | 9.001 | 699 |
| Overhead | 0.9% | 2.1% |
| <u>Processor 1</u> | | |
| J73I Programs | | 172 |
| Assembly Language Programs | 17.903 | 200 |
| Tables | | 252 |
| Total | 17.903 | 624 |
| Overhead | 1.8% | 1.9% |

3.6.3.1 DAC Tuning Approach

All frequently called procedures were recoded in assembly language. The calls to procedures RETCOR and GETCOR in MSM were coded inline as simple stacking/unstacking operations (all dynamic storage blocks being the same size). Internal procedure CPYD was coded inline

as a MOV instruction. The access functions required were BWD, ESWS, BSRS, CSTN, CST, USTN, UST and UDT.

The processing of access lists was handled in a manner similar to the handling of the active node stack in ITC. BACCSS and EACCSS start the processing by transferring to the begin or end access procedure respectively for the first access controller. Each access procedure then transfers control to the appropriate access procedure for the next controller on the list. Each list is terminated by a dummy access controller which causes control to be returned directly to the caller of BACCSS or EACCSS.

The DAC owned data structures were not modified.

3.6.3.2 Functional Differences with Baseline

- Only the functions BACCSS, EACCSS, BARD, ERS, BWD, BSRS, ESWS, CSTN, CST, USTN, UST and UDT are implemented.

3.6.3.3 DAC Final Tuned Detailed Timing Statistics

3.6.3.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|-------------------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| BACCSS | 8.6 | 224 | 1.926 |
| EACCSS | 8.6 | 164 | 1.410 |
| USTN | 15.4 | 151 | 2.325 |
| UST | 11.8 | 10 | 0.118 |
| UDT | 16.2 | 32 | 0.518 |
| CSTN | 17.6 | 8 | 0.140 |
| CST | 14.4 | 25 | 0.360 |
| BWD | 18.2 | 32 | 0.582 |
| ESWS | 15.6 | 34 | 0.530 |
| BSRS | 17.0 | 10 | 0.170 |
| Words copied via MOV instruction | 2.0 | 401 | <u>0.922</u> |
| | | Total | 9.001 |
| | | Overhead | 0.9% |

3.6.3.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| BACCSS | 8.6 | 227 | 1.952 |
| EACCSS | 8.6 | 225 | 1.935 |
| USTN | 15.4 | 103 | 1.586 |
| UST | 11.8 | 5 | 0.059 |
| BARD | 16.6 | 352 | 5.843 |
| ERD | 17.0 | 352 | 5.984 |
| ESWS | 13.6 | 40 | <u>0.544</u> |
| | | Total | 17.903 |
| | | Overhead | 1.8% |

3.6.3.4 DAC Final Tuned Detailed Space Statistics

3.6.3.4.1 Processor 0

| | Words |
|---------------------------|-----------|
| J73I Programs | |
| DACINT | 150 |
| Program Data Space | <u>22</u> |
| | 172 |
| Assembler Programs | |
| BACCSS | 8 |
| EACCSS | 8 |
| BARD | 16 |
| EARD | 22 |
| CST | 14 |
| USTN | 14 |
| UST | 12 |
| UDT | 14 |
| BWD | 18 |
| BRS | 16 |
| EWS | 14 |
| CSTN | 16 |
| Program Data Space | <u>28</u> |
| | 200 |

Data

| | |
|------------------|------------|
| Same as baseline | <u>327</u> |
| Total | 699 |
| Overhead | 2.1% |

3.6.3.4.2 Processor 1

Words

J73I Programs

| | |
|---------------------|-----|
| Same as processor 0 | 172 |
|---------------------|-----|

Assembler Programs

| | |
|---------------------|-----|
| Same as processor 0 | 200 |
|---------------------|-----|

Data

| | |
|------------------|------------|
| Same as baseline | <u>252</u> |
| Total | 624 |
| Overhead | 1.9% |

3.6.3.5 DAC Sensitivity Analysis

The time overhead in the final tuned version of DAC is about 30% of that in the baseline version. Copying of data in static storage blocks accounts for 10% of DAC overhead in processor 0 and 5% in processor 1. In both processors the rates at which static data is copied is relatively low. If the system response requirement and sharing of data were such that 400 static blocks averaging 10 words had to be copied every second, then the DAC overhead in processor 0 would be doubled.

3.6.4 SCH Final Tuned Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 16 |
| Assembly Language Programs | 8.330 | 44 |
| Tables | | |
| Total | 8.330 | 60 |
| Overhead | 0.8% | 0.2% |
| <u>Processor 1</u> | | |
| J73I Programs | | 16 |
| Assembly Language Programs | 12.141 | 44 |
| Tables | | |
| Total | 12.141 | 60 |
| Overhead | 1.2% | 0.2% |

3.6.4.1 SCH Tuning Approach

The outside calls to procedures DROP and AUTASK were recoded as inline assembler code. The spare time computing procedure (SPRTIM) was dropped since it is not required in a checked out system. SCHED was coded in assembler language thus eliminating calls to TGTR and TSUM.

Data structure used by SCH were not modified.

3.6.4.2 Functional Differences

- Spare task execution time is not computed.
- Overload flag is not used.

3.6.4.3 SCH Final Tuned Detailed Timing Statistics

3.6.4.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| SCHED entries | 28.4 | 157 | 4.459 |
| Active tasks not preemptable | 9.8 | 157 | 1.359 |
| More urgent tasks | 8.0 | 314 | <u>2.512</u> |
| | | Total | 8.330 |
| | | Overhead | 0.8% |

3.6.4.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| SCHED entries | 28.4 | 224 | 6.362 |
| Active tasks not preemptable | 9.8 | 224 | 2.195 |
| More urgent tasks | 8.0 | 448 | <u>3.584</u> |
| | | Total | 12.141 |
| | | Overhead | 1.2% |

3.6.4.4 SCH Final Tuned Detailed Space Statistics

3.6.4.4.1 Processor 0

| | Words |
|---------------------------|-----------|
| J73I Programs | |
| SCHINT | 12 |
| Program Data Space | <u>4</u> |
| | 16 |
| Assembler Programs | |
| SCHEDE | <u>44</u> |
| | 44 |
| Data | |
| None | — |
| Total | 60 |
| Overhead | 0.2% |

3.6.4.4.2 Processor 1

| | Words |
|---------------------------|-------|
| J73I Programs | |
| Same as processor 0 | 16 |
| Assembler Programs | |
| Same as processor 0 | 44 |
| Data | |
| None | — |
| Total | 60 |
| Overhead | 0.2% |

3.6.4.5 SCH Sensitivity Analysis

The time overhead in the final tuned version of SCH is approximately 25% of the untuned version. In the tuned version as in the baseline version, loop setup and actual insertion accounts for approximately 50% of the overhead. The overhead is directly proportional to the number of tasks scheduled per second times the average number of tasks which are in the queue and are more urgent. In contrast to the clock queue which gets longer with each additional clock, the scheduler queue length tends to remain nearly empty. Tasks are removed from the queue as fast as they are inserted except at brief beats where several tasks are scheduled nearly simultaneously.

3.6.5 DTF Final Tuned Statistics

| Processor | Time/second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| Processor 0 | | |
| J73I Programs | 0 | 270 |
| Assembly Language Programs | 39,354 | 1051 |
| Tables | | 1268 |
| Total | 39,354 | 2589 |
| Overhead | 3.9% | 7.9% |
| Processor 1 | | |
| J73I Programs | 0 | 300 |
| Assembly Language Programs | 32,306 | 1139 |
| Tables | | 643 |
| Total | 32,306 | 2082 |
| Overhead | 3.2% | 6.4% |

3.6.5.1 Tuning Approach

DTF tuning consisted of deleting the generalities associated with dynamic subaddress allocation, priority, bus control passing, and dynamic storage when they were not required, including ENABLE, DISABL, QUEUE, DQUEUE, GETCOR and RETCOR inline and straight line coding of the loops in DTFPAS. Space was reduced additionally by removing head only code in processor 1 and non-head only code in processor 0.

3.6.5.2 Functional Differences with Baseline

The following are differences between the baseline executive and the final tuned executive in processor 0:

- Control is passed on a round robin basis.
- Only one remote receive subaddress field.
- One dummy word is transmitted when control is passed.
- Dynamic storage for remote receive not supported.
- Unallocated subaddresses for master receive not supported.

The same differences occur in processor 1. The following additional differences exist:

- Dynamic/unallocated subaddresses for master transmit not supported.
- Dynamic storage remote receive is supported.

3.6.5.3 DTF Final Tuned Detailed Timing Statistics

3.6.5.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|--|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| DTFACT | | | |
| ITCACT to be queued | 21.6 | 90 | 1.944 |
| No ITCACT notification req. | 16.0 | 10 | .160 |
| Dynamic storage data transmitted | 23.8 | 32 | <u>.762</u> |
| | | | 2.866 |
| DTFPAS | | | |
| Each entry | 26.4 | 100 | 2.640 |
| ITCPAS to be queued | 9.2 | 5 | .046 |
| Each block of data received | 8.8 | 5 | <u>.044</u> |
| | | | 2.730 |
| DTFKEY | 29.0 | 100 | 2.900 |
| SENDYN | 29.0 | 32 | .928 |
| SEND | | | |
| Allocated entry | 28.8 | 195 | 5.620 |
| Allocated additional command word/entry | 27.2 | 262 | 7.130 |
| Allocated notify | 9.6 | 195 | 1.870 |
| Unallocated entry | 48.8 | 40 | 1.952 |
| Unallocated additional Command word/entry | 47.2 | 10 | .472 |
| Unallocated notify | 9.6 | 20 | .190 |
| | | | 17.170 |
| BCI2 | | | |
| Take control of bus | 79.2 | 100 | 7.920 |
| Relinquish control of bus | 48.4 | 100 | 4.840 |
| | | | <u>12.760</u> |
| Total | | 39.354 | |
| Overhead | | | 3.9% |

3.6.5.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /second | Total (milliseconds) |
|---|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| DTFACT | | | |
| ITCACT to be queued | 15.6 | 60 | .936 |
| No ITCACT notification required | 10.0 | 40 | .400 |
| | | | 1.336 |
| DTFPAS | | | |
| Each entry | 73.4 | 100 | 7.340 |
| ITCPAS to be queued | 9.2 | 50 | .460 |
| Each block of statically allocated data received | 8.8 | 5 | .044 |
| Each block of dynamically allocated data received | 19.6 | 60 | 1.176 |
| | | | 9.020 |
| DTFKEY | 15.8 | 100 | 1.580 |
| SEND | | | |
| Each entry | 25.2 | 110 | 2.770 |
| Each additional command word/entry | 23.6 | 165 | 3.890 |
| Each notify | 9.2 | 103 | .950 |
| | | | 7.610 |
| BCI2 | | | |
| Take control of bus | 79.2 | 100 | 7.920 |
| Relinquish control of bus | 48.4 | 100 | 4.840 |
| | | | <u>12.760</u> |
| | | Total | 32.306 |
| | | Overhead | 3.2% |

3.6.5.4 DTF Final Tuned Detailed Space Requirements

3.6.5.4.1 Processor 0

| | Words |
|---------------------------------|-------------|
| J73/I Programs | |
| DTFINT | 102 |
| FLIP | 168 |
| | 270 |
| Assembler Programs | |
| SEND | 70 |
| SENDYN | 26 |
| DTFPAS | 86 |
| DTFACT | 46 |
| DTFKEY | 32 |
| BC1U Initialization | 70 |
| BC1U interrupt handlers | 721 |
| | 1051 |
| Data | |
| DEVTBL | 36 |
| CWTBL | 636 |
| CWQCWP | 41 |
| MNOTFY | 64 |
| RNOTFY | 16 |
| R CODE | 5 |
| FLPDEV | 40 |
| Miscellaneous Items J73I | 8 |
| Subaddress Pointer Words | 112 |
| Storage for Executive 'Signals' | 6 |
| Scratch Storage | 32 |
| Miscellaneous Items Assembler | 32 |
| Command Word Storage | 240 |
| | <u>1268</u> |
| Total | 2589 |
| Overhead | 7.9% |

3. 6. 5. 4. 2 Processor 1

| | Words |
|---------------------------------|-------------------|
| J73/I Programs | |
| DTFINT | 132 |
| FLIP | 168 |
| | 300 |
| Assembler Programs | |
| SEND | 34 |
| DTFACT | 16 |
| DTFPAS | 274 |
| DTFKEY | 20 |
| BC1U Initialization | 42 |
| BC1U interrupt handlers | 753 |
| | 1139 |
| Data | |
| DEVTBL | 40 |
| CWTBL | 164 |
| MNOTFY | 16 |
| RNOTFY | 64 |
| RCODE | 15 |
| FLPDEV | 30 |
| Misc Items J73I | 8 |
| Subaddress Pointer Words | 92 |
| Storage for Executive 'Signals' | 12 |
| Scratch Storage | 32 |
| Misc Items Assembler | 30 |
| Command Word Storage | 140 |
| | <u>643</u> |
| Total | 2082 |
| Overhead | 6. 4% |

3.6.5.5 DTF Sensitivity Analysis

The time overhead in the final tuned version of the DTF is approximately 45% of the processor 0 baseline and 35% of the processor 1 baseline. The time reduction for the final tuned DTF over the baseline was less dramatic than for other function clusters because a significant portion of the baseline was already in assembly language. The final tuned assembly language portion was over 72% of the time overhead of the baseline version with the reduction due almost entirely to a reduction in generality.

The time overhead of the final tuned version is sensitive to the same parameters as the baseline version, namely, the number of times control is passed per second (BCI2) and the number of different interprocessor data blocks that can be received (DTFPAS).

An alternative approach to DTFPAS may be desirable for executives where the number of different interprocessor data blocks is large (on the order of twenty or more).

DTFPAS currently has all remote receive (interprocessor) tagwords as zero prior to entering remote mode. DTFPAS examines each tagword for non-zero (data received) each time another processor relinquishes master control. An alternative approach would have the processor in master control allocate a sequential subaddress, starting at one, for each processor to processor transmission. This would allow DTFPAS to terminate the tagword examination on encountering the first non-zero tagword. DTFPAS overhead would be dramatically reduced for large numbers of different interprocessor data blocks, with increased overhead in dealing with dynamic storage and extracting the data block identification from the data.

3.6.6 MSM Final Tuned Statistics

| Processor | Time/second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 0 |
| Assembly Language Programs | | 14 |
| Tables | | 69 |
| Total | 0 | 83 |
| Overhead | 0% | .2% |
| <u>Processor 1</u> | | |
| J73I Programs | | 0 |
| Assembly Language Programs | | 14 |
| Tables | | 715 |
| Total | 0 | 729 |
| Overhead | 0% | 2.2% |

3.6.6.1 Functional Differences with Baseline

The tuned version on each processor is capable of allocating only blocks 34 words long.

3.6.6.2 Tuning Approach

The cluster maintains a list of 34 word blocks. Each time a block is allocated, it is removed from the front of the list; when returned, it is placed at the front of the list.

The code to perform these activities has been placed inline in all places where calls are required, except in initialization. The time and space for these has been included in the statistics for the cluster manipulating the storage. Each inline call to GETCOR is 6.2 microseconds and to RETCOR is 6.4 microseconds yielding .04% overhead in processor 0 and .08% overhead in processor 1.

3.6.6.3 MSM Final Tuned Detailed Timing Statistics

3.6.6.3.1 Processor 0

The times have been included in the calling cluster.

3.6.6.3.2 Processor 1

The times have been included in the calling cluster.

3.6.6.4 MSM Final Tuned Detailed Space Statistics

3.6.6.4.1 Processor 0

Words

Assembler Programs

| | |
|--------|----|
| MSMINT | 14 |
|--------|----|

The GETCOR and RETCOR space has
been included in the calling clusters.

Data

| | |
|------------|-----------|
| Two blocks | <u>69</u> |
| | 83 |

3.6.6.4.2 Processor 1

Words

Assembler Programs

| | |
|--------|----|
| MSMINT | 14 |
|--------|----|

The GETCOR and RETCOR space has
been included in the calling clusters.

Data

| | |
|-------------------|------------|
| Twenty-one blocks | <u>715</u> |
| | 729 |

3.6.6.5 Sensitivity Analysis

The sequence of calls does not affect timing since blocks are always removed from the front of the list and placed on the front when returned. However, the call sequence does affect space since there must be enough blocks to satisfy all outstanding requests.

3.6.7 DSP Final Tuned Statistics

| Processor | Time/Second (milliseconds) | Space (words) |
|-------------------------------|-------------------------------|------------------|
| <u>Processor 0</u> | | |
| J73I Programs | | 92 |
| Assembly Language Programs | 13.811 | 26 |
| Tables | | 69 |
| Total | 13.811 | 187 |
| Overhead | 1.4% | 0.6% |
| <u>Processor 1</u> | | |
| J73I Programs | | 92 |
| Assembly Language Programs | 14.982 | 26 |
| Tables | | 69 |
| Total | 14.982 | 187 |
| Overhead | 1.5% | 0.6% |

3.6.7.1 DSP Tuning Approach

A more efficient interrupt/return linkage was devised. Register 14 was dedicated to the save area/stack pointer (CURSAV). When the processor is in the application, register 14 points to the top of the stack. When the executive is entered (either via an interrupt or when an application task completes), register 14 is complemented. Hence, when an interrupt is taken, the interrupt handler can immediately determine whether the processor was in the task or executive state by testing the sign of register 14 imposed by the baseline executive. Use of register 14 also eliminates the restriction in memory layout and thereby avoids saving registers when in the executive state. When the executive is exited via DSPTSK, register 14 is again complemented to indicate return to the task state.

The interface functions ENABLE, DISABL, QUEUE and ENDTSK were eliminated in favor of inline code in all calling procedures.

3.6.7.2 Functional Differences

- None.

3.6.7.3 DSP Final Tuned Timing Statistic

3.6.7.3.1 Processor 0

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| QUEUE | | | |
| Each entry | | | |
| Task ends | | 157 | |
| DTF interrupt returns | | 200 | |
| TIM interrupt returns | | 90 | |
| | 6.8 | <u>447</u> | <u>3.040</u> |
| Each item queued | | | |
| DTF queuing | | 95 | |
| Interrupts in exec | | 29 (10% of 290) | |
| | 6.8 | <u>124</u> | <u>0.843</u> |
| | | | 3.883 |
| DSPTSK | | | |
| Interrupt returns | 7.2 | 290 | 2.088 |
| Task starts | 7.8 | 157 | <u>1.225</u> |
| | | | <u>3.313</u> |
| RSTART | 20.0 | 290 | 5.800 |
| START | 11.2 | 157 | <u>1.758</u> |
| | | Total | 14.75 |
| | | Overhead | 1.5% |

3.6.7.3.2 Processor 1

| Procedure | Processing Time (microseconds) | Occurrences /Second | Total (milliseconds) |
|---------------------------|-----------------------------------|------------------------|-------------------------|
| Assembler Programs | | | |
| DQUEUE | | | |
| Each entry | | | |
| Task ends | 224 | | |
| DTF interrupt returns | 200 | | |
| TIM interrupt returns | 67 | | |
| | 6.8 | 491 | 3.339 |
| Each item queued | | | |
| DTF queueing | 110 | | |
| Interrupts in exec | 27 (10% of 267) | | |
| | 6.8 | 137 | 0.952 |
| | | | 4.271 |
| DSPTSK | | | |
| Interrupt returns | 7.2 | 267 | 1.922 |
| Task starts | 7.8 | 224 | 1.747 |
| | | | 3.669 |
| RSTART | 20.0 | 267 | 5.340 |
| START | 11.2 | 224 | 2.509 |
| | | Total | 15.789 |
| | | Overhead | 1.6% |

3.6.7.4 DSP Final Tuned Detailed Space Statistics

3.6.7.4.1 Processor 0

| | Words |
|---------------------------|-----------|
| J73I Programs | |
| DSPINT | 78 |
| Program Data Space | <u>14</u> |
| | 92 |
| Assembler Programs | |
| DQUEUE | 6 |
| DSPTSK | 4 |
| RSTART | 6 |
| START | <u>10</u> |
| | 26 |
| Data | |
| Same as baseline | <u>69</u> |
| Total | 187 |
| Overhead | 0.6% |

3.6.7.4.2 Processor 1

| | Words |
|---------------------------|-----------|
| J73I Programs | |
| Same as processor 0 | 92 |
| Assembler programs | |
| Same as processor 0 | 26 |
| Data | |
| Same as baseline | <u>69</u> |
| Total | 187 |
| Overhead | 0.6% |

3.6.7.5 DSP Sensitivity Analysis

The time overhead for the final tuned version of DSP is approximately 37% of the baseline version's overhead. Since parts of the dispatchers were already hand coded in assembler language, the improvement in speed is less marked than that for other clusters. The dispatcher overhead depends primarily on the number of interrupts and the number of task starts per second. Each additional 100 interrupts/second consumes about 0.4% of processor time. Each additional 100 task starts per second consumes about 0.3% of processor time.

3.6.8 SSM Final Tuning Statistics

Secondary Storage Management is not implemented.

3.6.9 MPL Final Tuning Statistics

Multiprocessor Locking is not implemented.

3.6.10 Final Tuned Bus Traffic

The only difference between the baseline and tuned bus control is that only one data word is required to pass control. The effect of this reduced bus overhead by 0.4%.

3.7 Conclusions

The approach taken allowed small and efficient baseline executives to be written. These were then tuned significantly for the DAIS mission. The executives were tuned by approximately 72% for time and 47% for space. The following subsections present the time and space statistics by function cluster and tuning method.

3.7.1 Statistical Reduction Allocated by Function Cluster

This section provides a series of figures which illustrate the time and space overhead allocated to each function cluster, and the overhead reduction between the baseline and tuned executives. Figures 6 and 7 shows the processor 0 time and space overhead, and Figures 8 and 9 show the time and space overhead for processor 1.

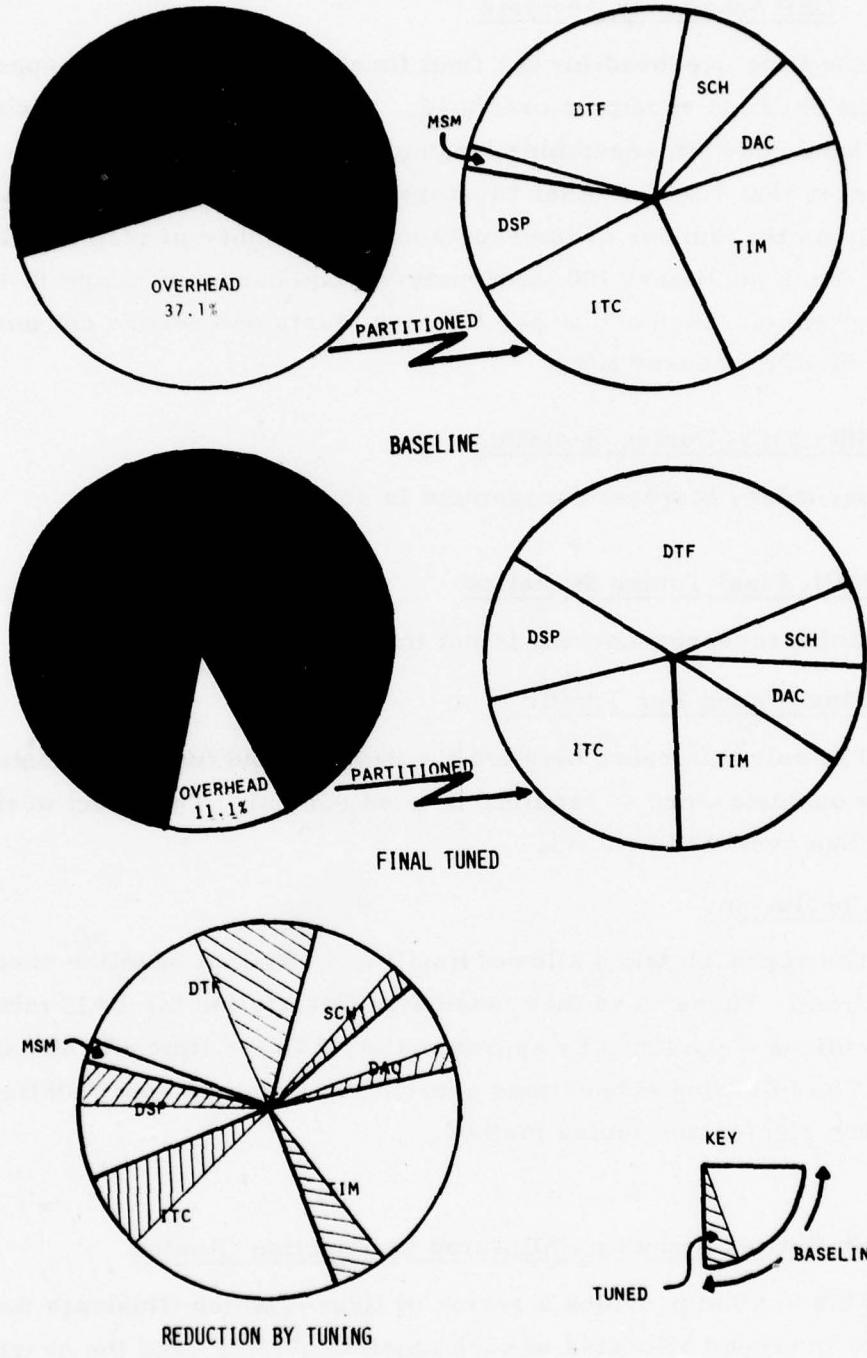


Figure 6. PROCESSOR 0 TIME OVERHEAD

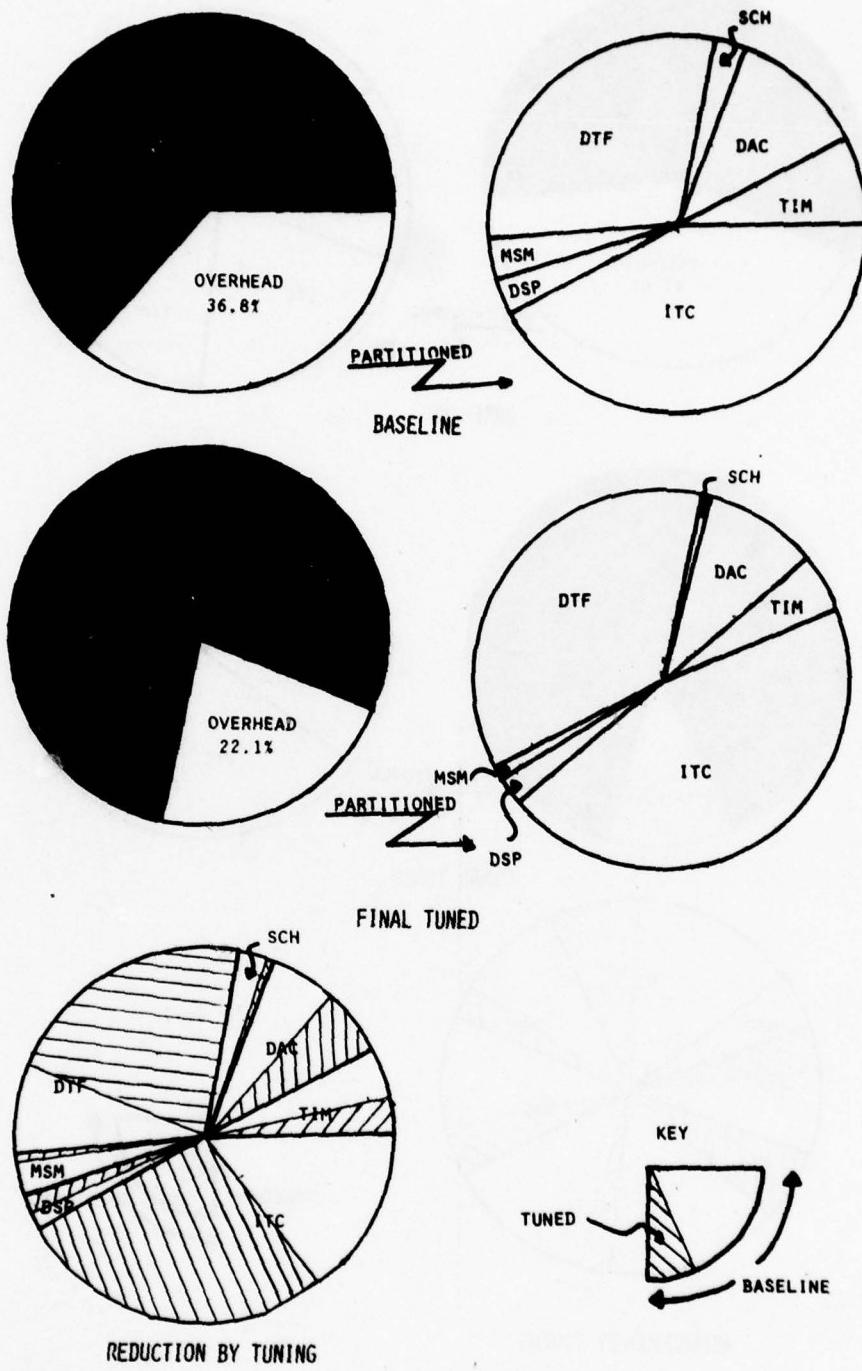
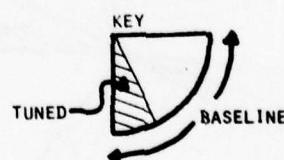
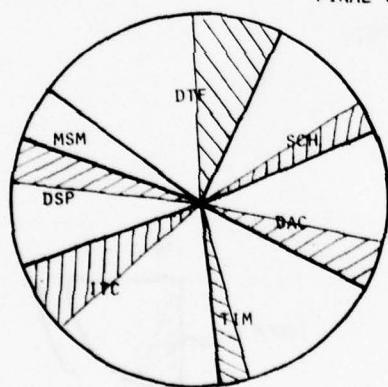
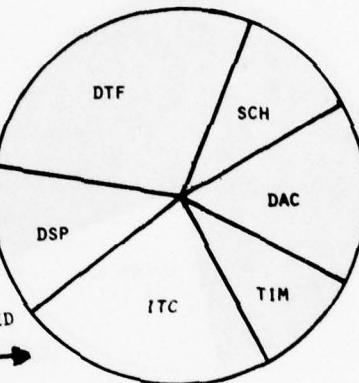
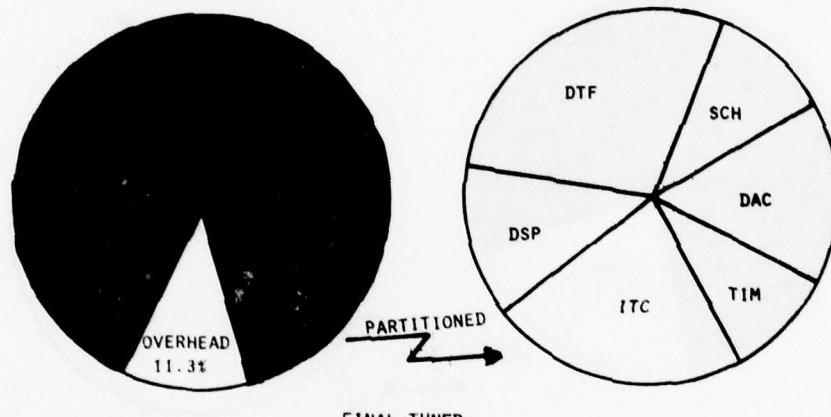
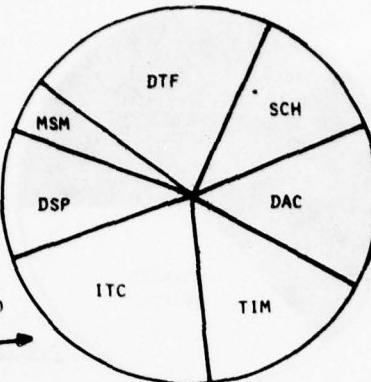
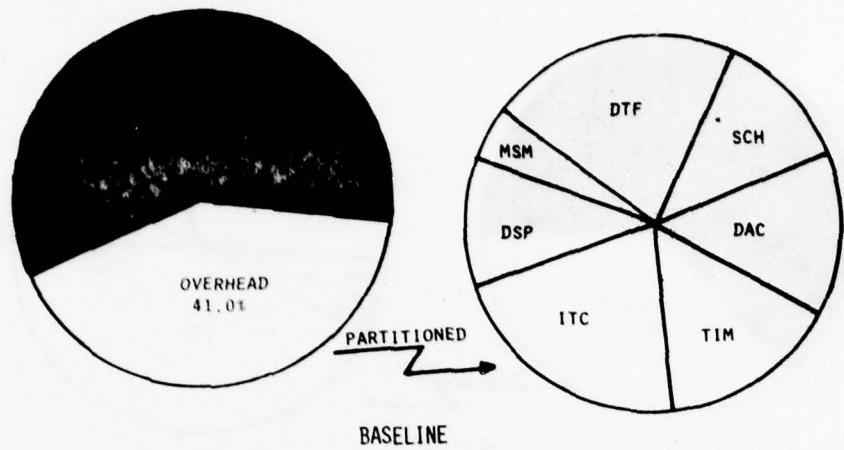


Figure 7. PROCESSOR 0 SPACE OVERHEAD



REDUCTION BY TUNING

Figure 8. PROCESSOR 1 TIME OVERHEAD

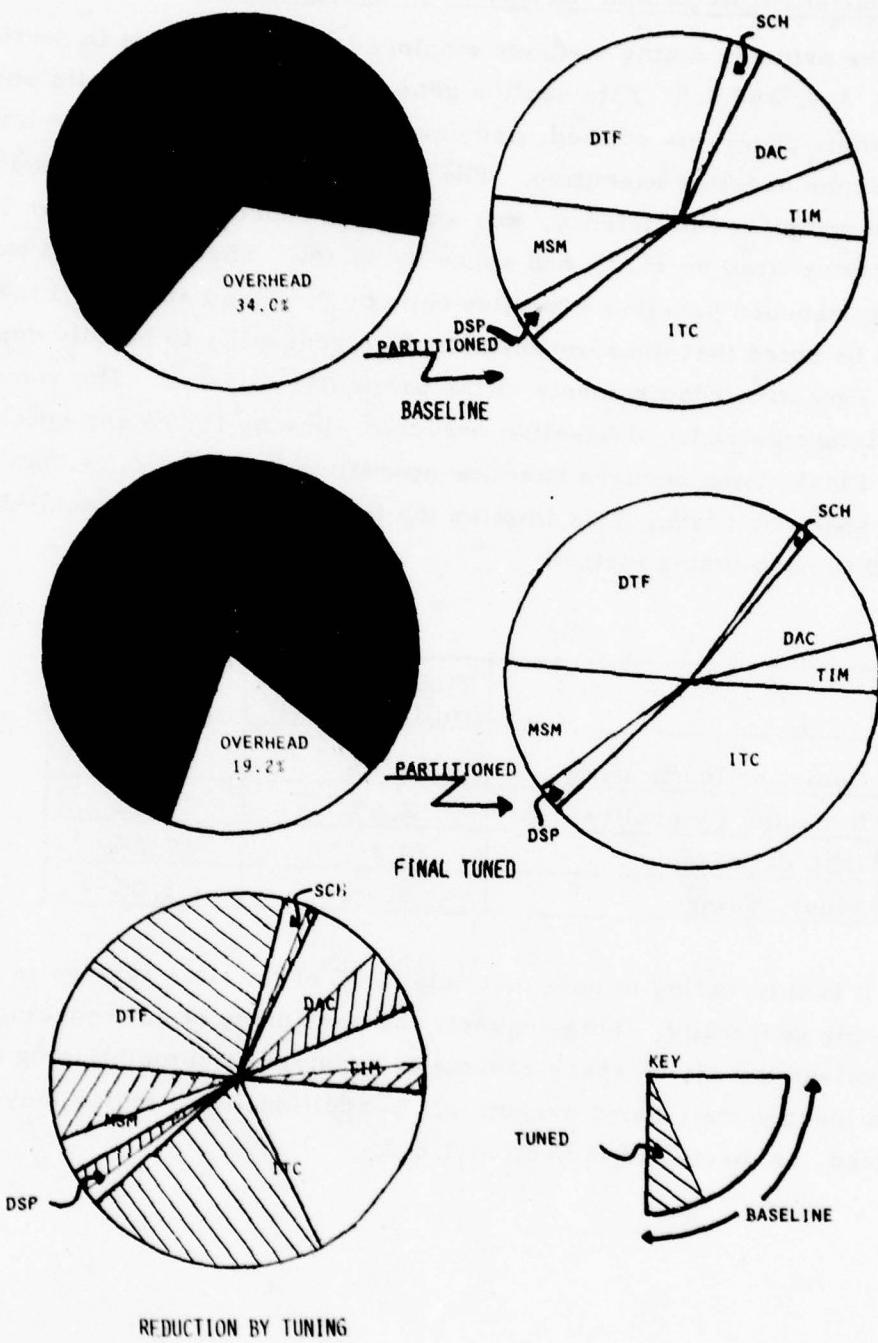


Figure 9. PROCESSOR 1 SPACE OVERHEAD

3.7.2 Statistical Reduction Allocated by Tuning Method

The primary tuning methods employed were described in Sections 3.2, 3.3, 3.4, and 3.5. This section generalizes from the results obtained in the sample programs studied, and uses the statistics to find the improvement over the baseline executive. The first tuning method, hand coding to eliminate compiler inefficiency, was shown to reduce the processor 0 baseline executive time by 48.2% and space by 40.0%. The removal of excess generality reduced baseline executive time by 2.0% and space by 13.4%. It should be noted that the removal of excess generality is heavily dependent upon the executive requirements of the particular mission. The removal of HOL deficiencies reduced baseline executive time by 16.9% and space by 11.0%. Final tuning reduced baseline sparetim overhead 2.7%, but increases space by 1.9%. This implies the following percent of total reduction allocated to each tuning method:

| | Time/second (milliseconds) | Space (words) |
|-----------------------|-------------------------------|------------------|
| Compiler inefficiency | 69.1% | 64.0% |
| Reducing generality | 2.9% | 21.4% |
| HOL deficiency | 24.1% | 17.6% |
| Final Tuning | 3.9% | - 3.0% |

It is interesting to note that only 2.9% of the time savings is due to reducing generality. This suggests the possibility of not reducing generality (unless the 21.4% space savings is required) and maintaining a mission independent, tuned executive. In addition, final tuning may not be desired, as discussed in Section 3.5.2.

Appendix A

EXECUTIVE TUNING EXAMPLES

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A.1 CLOCKQ - TUNED FOR HOL INEFFICIENCY

| CLOCKQ | EQU | S | |
|--------|----------|------------------|-----------------------------|
| | STM | R5,CLOCSV | . 5.4 |
| | JS | R2,ENABLE | . 2.0 |
| | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| | DST | R0,PARM2 | . 2.6 |
| CLOC05 | L | R0,TIM'NEXT,R3 | . 2.0 |
| | ST | R0,TIM'NEXT | . 2.2 |
| | L | R0,TIM'EVENT,R3 | . 2.0 |
| | JC | EZ,CLOC10 | . 1.6/2.0 |
| | ST | R0,PARM1 | . 2.2 |
| | LIM | R15,PARML1 | . 1.6 |
| | JS | R2,SIGNAL | . 2.0 |
| CLOC10 | DL | R0,TIM'PERIOD,R3 | . 2.4 |
| | DC | R0,TIME0 | . 2.8 |
| | JC | EQ,CLOC20 | . 1.6/2.0 |
| | DST | R0,PARM1 | . 2.6 |
| | LIM | R15,PARML2 | . 1.6 |
| | JS | R2,TSUM | . 2.0 |
| | DL | R0,PARM3 | . 2.4 |
| | DST | R0,TIM'DUE,R3 | . 2.6 |
| ZR | R4 | | . 1.4 |
| CLOC15 | LR | R5,R4 | . 1.4 |
| | L | R4,TIM'NEXT,R5 | . 2.0 |
| | LIM | R0,TIM'DUE,R4 | . 1.6 |
| | ST | R0,PARML4 | . 2.2 |
| | LIM | R15,PARML3 | . 1.6 |
| | JS | R2,TGTR | . 2.0 |
| | L | R0,PARM1 | . 2.0 |
| | JC | GZ,CLOC15 | . 1.6/2.0 |
| | ST | R4,TIM'NEXT,R3 | . 2.2 |
| | ST | R3,TIM'NEXT,R5 | . 2.2 |
| CLOC20 | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| | DC | R0,PARM2 | . 2.8 |
| | JC | EQ,CLOC05 | . 1.6/2.0 |
| | DST | R0,PARM2 | . 2.6 |
| | LIM | R15,PARML2 | . 1.6 |
| | JS | R2,SETCLOCK | . 2.0 |
| | L | R0,PARM1 | . 2.0 |
| | JC | EZ,CLOC05 | . 1.6/2.0 |
| | LIM | R5,CLOCSV | . 5.8 |
| | J | R0,R2 | . 2.0 |
| PARML1 | CONSTANT | PARM1 | |
| PARML2 | CONSTANT | PARM2 | |
| | CONSTANT | PARM1 | |
| PARML3 | CONSTANT | PARM3 | 92.0 EACH ENTRY |
| PARML4 | STORAGE | 1 | 60.0 EACH CLOCK W/SAME TIME |
| | CONSTANT | PARM1 | AS PREVIOUS |
| | EVEN | | 14.6 EACH CLOCK ON QUEUE |
| PARM1 | STORAGE | 2 | W/EARLIER TIME |
| PARM2 | STORAGE | 2 | |
| PARM3 | STORAGE | 2 | |
| CLOCSV | STORAGE | 6 | |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A. 2 SEND - TUNED FOR HOL INEFFICIENCY

| SEND | EQU | \$ | | |
|--------|----------------|-------------------|---------------|-------|
| SEND05 | L | R15,0,R15 | . 2.0 | |
| | L | R3,0,R15 | . 2.0 | |
| | JC | EZ,SEND35 | . 1.6/2.0 | |
| | DST | R2,SENUSV | . 2.0 | |
| | JS | R2,DISABL | . 2.0 | |
| | L | R2,MCW | . 2.0 | |
| | AIM | R2,-2 | . 1.6 | |
| | ST | R2,MCW | . 2.2 | |
| | DL | R0,CW'RCV,R3 | . 2.4 | |
| | DST | R0,CwRCV,R2 | . 2.0 | |
| L | R15,CW'TYP,R3 | . 2.0 | | |
| L | R15,JPMTBL,R15 | . 2.0 | | |
| J | 0,R15 | . 2.0 | | |
| NTURRU | A | R0,URP1SA | . 2.0 | |
| | ST | R0,CWRCCV,R2 | . 2.2 | |
| | IM | URP1SA | . 3.2 | |
| | NTURRA | L | R15,USTSA | . 2.0 |
| CIM | | R15,IUDTSA | . 2.0 | |
| JC | | GE,SEND15 | . 1.6/2.0 | |
| AR | | R1,R15 | . 1.4 | |
| ST | | R1,CWTRA,R2 | . 2.2 | |
| IM | | USTSA | . 3.2 | |
| L | | R1,MHUFI | . 2.0 | |
| JC | | GZ,SEND10 | . 1.6/2.0 | |
| AIM | | R15,MTSAP2-MTSAPI | . 1.6 | |
| SEND10 | | L | R0,CWDADR,R3 | . 2.0 |
| | ST | R0,MTSAP1,R15 | . 2.2 | |
| | J | NOTIFY | . 2.0 | |
| | SEND15 | AIM | R2,2 | . 1.6 |
| ST | | R2,MCW | . 2.2 | |
| L | | R1,CWQPTR | . 2.0 | |
| ST | | R3,CwQ,R1 | . 2.2 | |
| IM | | CWQPTR | . 3.2 | |
| ZR | | R3 | . 1.4 | |
| J | | NOTIFY | . 2.0 | |
| NTARRU | | A | R0,URP1SA | . 2.0 |
| | | ST | R0,CWRCCV,R2 | . 2.2 |
| | | IM | URP1SA | . 3.2 |
| | J | NOTIFY | . 2.0 | |
| MRUDYN | L | R15,UDRSA | . 2.0 | |
| | AR | R0, R15 | . 1.4 | |
| | ST | R0,CWRCCV,R2 | . 2.2 | |
| | L | R0,MHUFI | . 2.0 | |
| | JC | EZ,SEND20 | . 1.6/2.0 | |
| | AIM | R15,MRSAP2-MRSAPI | . 1.6 | |
| | SEND20 | L | R0,MRSAP1,R15 | . 2.0 |
| | | ST | R0,Cw'DADR,R3 | . 2.2 |
| | | IM | UDRSA | . 3.2 |
| | | J | NOTIFY | . 2.0 |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A.2 SEND - TUNED FOR HOL INEFFICIENCY (Continued)

| | | | |
|--------|----------|-------------------|-----------------------------|
| MRUSTA | L | R15,USRSA | . 2.0 |
| | AR | R0,R15 | . 1.4 |
| | ST | R0,CWRCCV,R2 | . 2.2 |
| | IM | USRSA | . 3.2 |
| | L | R1,MBUFI | . 2.0 |
| | JC | GZ,SEND25 | . 1.6/2.0 |
| | AIM | R15,MRSAP2-MRSAP1 | . 1.6 |
| SEND25 | L | R0,CW*DADR,R3 | . 2.0 |
| | ST | R0,MRSAP1,R15 | . 2.2 |
| NOTIFY | L | R0,CW*CODE,R3 | . 2.0 |
| | JC | FZ,SEND30 | . 1.6/2.0 |
| | L | R2,MNINDEX | . 2.0 |
| | L | R1,CW*DADR,R3 | . 2.0 |
| | DST | R0,MNCODE,R2 | . 2.0 |
| | AIM | R2,2 | . 1.6 |
| | ANDM | R2,MNMAX | . 1.6 |
| | ST | R2,MNINDEX | . 2.2 |
| SEND30 | JS | R2,ENABLE | . 2.0 |
| | L | R3,CW*NXT,R3 | . 2.0 |
| | JC | NEZ,SEND05 | . 1.6/2.0 |
| | DL | R2,SENDSV | . 2.4 |
| SEND35 | J | 0,R2 | . 2.0 |
| * | | EVEN | |
| SENDSV | STORAGE | 2 | 41.0 USEC ALLOCATED ENTRY |
| | CONSTANT | MTURRU | 28.8 USEC ALLOC ADDITION CW |
| JMPTBL | CONSTANT | MTURRA | 64.2 USEC UNALLOCATED ENTRY |
| | CONSTANT | MTARRU | 52.0 USEC UNALLOC ADDIT CW |
| | CONSTANT | NOTIFY | 11.6 USEC NOTIFICATION |
| | CONSTANT | MRUDYN | |
| | CONSTANT | MRUSTA | |
| | CONSTANT | NOTIFY | |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A. 3 QUEUE/DQUEUE - TUNED FOR HOL INEFFICIENCY

| | | | |
|--------|-----------------|-----------|-----------------------------|
| QUEUE | EQU | \$ | |
| L | R15,0,R15 | . 2.0 | |
| L | R15,0,R15 | . 2.0 | |
| L | R1,EXCQ | . 2.0 | |
| ST | R1,EXC'NEXC,R15 | . 2.2 | |
| ST | R15,EXCQ | . 2.2 | |
| J | 0,R2 | . 2.0 | 12.4 USEC |
| DQUEUE | EQU | \$ | |
| ST | R2,DQUESV | . 2.2 | |
| DQLOOP | JS | R2,DISABL | . 2.0 |
| L | R1,EXCQ | . 2.0 | |
| JC | EZ,DQUE05 | . 1.6/2.0 | |
| L | R0,EXC'NEXC,R1 | . 2.0 | |
| ST | R0,EXCQ | . 2.2 | |
| LIM | R0,EXC'PRC,R1 | . 1.6 | |
| ST | R0,PARML1 | . 2.2 | |
| LIM | R15,PARML1 | . 1.6 | |
| JS | R2,CEXCPRC | . 2.0 | |
| J | DQLOOP | . 2.0 | |
| DQUE05 | JS | R2,DSPTSK | . 2.0 |
| | JI | DQUESV | |
| DQUESV | STORAGE | 1 | 10.2 USEC PLUS |
| PARML1 | STORAGE | 1 | 15.2 USEC EACH CEXCPRC CALL |

THIS PAGE IS BEST QUALITY PRACTICABLE
 FROM COPY FURNISHED TO DDC

A. 4 CLOCKQ - TUNED FOR EXCESS GENERALITY

| | | | |
|--------|----------|------------------|-----------------------------|
| CLOCKQ | EQU | \$ | |
| | STM | R5,CLOCsv | . 5.4 |
| | JS | R2,ENABLE | . 2.0 |
| | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| | DST | R0,PARM2 | . 2.6 |
| CLOCK5 | L | R0,TIM'NEXT,R3 | . 2.0 |
| | ST | R0,TIM'NEXT | . 2.2 |
| | L | R0,TIM'EVENT,R3 | . 2.0 |
| | JC | EZ,CLOC10 | . 1.6/2.0 |
| | ST | R0,PARM1 | . 2.2 |
| | LIM | R15,PARML1 | . 1.6 |
| | JS | R2,SIGNAL | . 2.0 |
| CLOC10 | LIM | R0,TIM'PERIOD,R3 | . 1.6 |
| | ST | R0,PARML5 | . 2.2 |
| | LIM | R15,PARML2 | . 1.6 |
| | JS | R2,TSUM | . 2.0 |
| | DL | R0,PARM3 | . 2.4 |
| | DST | R0,TIM'DUE,R3 | . 2.6 |
| | ZR | R4 | . 1.4 |
| CLOC15 | LR | R5,R4 | . 1.4 |
| | L | R4,TIM'NEXT,R5 | . 2.0 |
| | LIM | R0,TIM'DUE,R4 | . 1.6 |
| | ST | R0,PARML4 | . 2.2 |
| | LIM | R15,PARML3 | . 1.6 |
| | JS | R2,TGTR | . 2.0 |
| | L | R0,PARM1 | . 2.0 |
| | JC | GZ,CLOC15 | . 1.6/2.0 |
| | ST | R4,TIM'NEXT,R3 | . 2.2 |
| | ST | R3,TIM'NEXT,R5 | . 2.2 |
| CLOC20 | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| | DC | R0,PARM2 | . 2.8 |
| | JC | EQ,CLOC05 | . 1.6/2.0 |
| | DST | R0,PARM2 | . 2.6 |
| | LIM | R15,PARML2 | . 1.6 |
| | JS | R2,SETCLOCK | . 2.0 |
| | L | R0,PARM1 | . 2.0 |
| | JC | EZ,CLOC05 | . 1.6/2.0 |
| | LM | R5,CLOCsv | . 5.8 |
| | J | 0,R2 | . 2.0 |
| PARML1 | CONSTANT | PARM1 | |
| PARML2 | CONSTANT | PARM2 | |
| PARML5 | STORAGE | 1 | |
| PARML3 | CONSTANT | PARM3 | 86.4 EACH ENTRY |
| PARML4 | STORAGE | 1 | 55.0 EACH CLOCK W/SAME TIME |
| | CONSTANT | PARM1 | AS PREVIOUS |
| | EVEN | | 14.6 EACH CLOCK ON QUEUE |
| PARM1 | STORAGE | 2 | W/EARLIER TIME |
| PARM2 | STORAGE | 2 | |
| PARM3 | STORAGE | 3 | |
| CLOCsv | STORAGE | 6 | |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A.5 SEND - TUNED FOR EXCESS GENERALITY

| SEND | EQU | \$ | |
|--------|---------|-------------------|---|
| SEND05 | L | R15,0,R15 | . 2.0 |
| | L | R3,0,R15 | . 2.0 |
| | JC | EZ,SEND25 | . 1.6/2.0 |
| | DST | R2,SENDSV | . 2.6 |
| | JS | R2,DISABL | . 2.0 |
| | L | R2,MCW | . 2.0 |
| | AIM | R2,-2 | . 1.6 |
| | ST | R2,MCW | . 2.2 |
| | DL | R0,CW'RCV,R3 | . 2.4 |
| | DST | R0,CWRFCV,R2 | . 2.6 |
| SEND10 | L | R15,CW'TYP,R3 | . 2.0 |
| | JC | NEZ,NOTIFY | . 1.6/2.0 |
| | L | R15,USTSA | . 2.0 |
| | CIM | R15,IUDTSA | . 2.0 |
| | JC | LT,SEND10 | . 1.6/2.0 |
| | AIM | R2,2 | . 1.6 |
| | ST | R2,MCW | . 2.2 |
| | L | R1,CWQPTR | . 2.0 |
| | ST | R3,CWQ,R1 | . 2.2 |
| | IM | CWQPTR | . 3.2 |
| SEND15 | ZR | R3 | . 1.4 |
| | J | NOTIFY | . 2.0 |
| | AR | R1,R15 | . 1.4 |
| | ST | R1,CWTTRA,R2 | . 2.2 |
| | IM | USTSA | . 3.2 |
| SEND20 | L | R1,MBUF1 | . 2.0 |
| | JC | GZ,SEND15 | . 1.6/2.0 |
| | ATM | R15,MTSAP2-MTSAP1 | . 1.6 |
| | L | R0,CW'DADR,R3 | . 2.0 |
| | ST | R0,MTSAP1,R15 | . 2.2 |
| SEND25 | NOTIFY | L R0,CW'CODE,R3 | . 2.0 |
| | JC | EZ,SEND20 | . 1.6/2.0 |
| | L | R2,MNINDX | . 2.0 |
| | L | R1,CW'DADR,R3 | . 2.0 |
| | DST | R0,MNCODE,R2 | . 2.6 |
| | AIM | R2,2 | . 1.6 |
| | ANOM | R2,MNMAX | . 1.6 |
| | ST | R2,MNINDX | . 2.2 |
| | JS | R2,ENABLE | . 2.0 |
| | L | R3,CW'NXT,R3 | . 2.0 |
| SENDSV | JC | NEZ,SEND05 | . 1.6/2.0 |
| | DL | R2,SENDSV | . 2.4 |
| | J | 0,R2 | . 2.0 |
| | EVEN | | |
| SENDSV | STORAGE | 2 | 38.6 USEC ALLOCATED ENTRY 26.8 USEC ALLOC ADDITION CW 59.8 USEC UNALLOCATED ENTRY 48.0 USEC UNALLOC ADDIT CW 11.6 USEC NOTIFICATION |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A. 6 CLOCKQ - TUNED FOR HOL DEFICIENCIES

| | | | |
|---------------|-----------------|-------------------------|------------------------------------|
| CLOCKQ | EQU | \$ | |
| | STM | R5,CLOCsv | . 5.4 |
| | ENBL | | |
| | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| CLOC02 | DST | R0,PARM2 | . 2.6 |
| CLOC05 | L | R0,TIM'NEXT,R3 | . 2.0 |
| | ST | R0,TIM'NEXT | . 2.2 |
| | L | R0,TIM'EVENT,R3 | . 2.0 |
| | JC | EZ,CLOC10 | . 1.6/2.0 |
| | ST | R0,PARM1 | . 2.2 |
| | LIM | R15,PARML1 | . 1.6 |
| | JS | R2,SIGNL | . 2.0 |
| CLOC10 | DL | R0,TIM'PERIOD,R3 | . 2.4 |
| | DA | R0,PARM2 | |
| | DST | R0,TIM'DUE,R3 | . 2.6 |
| | ZR | R4 | . 1.4 |
| CLOC15 | LR | R5,R4 | . 1.4 |
| | L | R4,TIM'NEXT,R5 | . 2.0 |
| | DC | R0,TIM'DUE,R4 | |
| | JC | GT,CLOC15 | . 1.6/2.0 |
| | ST | R4,TIM'NEXT,R3 | . 2.2 |
| | ST | R3,TIM'NEXT,R5 | . 2.2 |
| CLOC20 | L | R3,TIM'NEXT | . 2.0 |
| | DL | R0,TIM'DUE,R3 | . 2.4 |
| | DC | R0,PARM2 | . 2.8 |
| | JC | EQ,CLOC05 | . 1.6/2.0 |
| | ITB | R2 | |
| | SK | R2,R1 | |
| | JC | GEZ,CLOC02 | |
| | MSIM | R2,10 | |
| | OTA | R2 | |
| | LM | R5,CLOCsv | . 5.8 |
| | J | U,R2 | . 2.0 |
| . | | | |
| . | | | |
| . | | | |
| . | | | |
| . | | | |
| PARML1 | CONSTANT | PARM1 | 58.4 EACH ENTRY |
| | CONSTANT | PARM2 | 38.6 EACH CLOCK W/SAME TIME |
| | EVEN | | AS PREVIOUS |
| PARM1 | STORAGE | 2 | 5.4 EACH CLOCK ON QUEUE |
| PARM2 | STORAGE | 2 | W/EARLIER TIME |
| CLOCsv | STORAGE | 6 | |

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

A. 7 SEND - TUNED FOR HOL DEFICIENCIES

| SEND | EQU | \$ | |
|--------|---------|------------------|------------|
| | L | R15,W,R15 | . 2.0 |
| | L | R15,W,R15 | . 2.0 |
| | JC | EZ,SEND25 | . 1.6/2.0 |
| | ST | R2,SENDSV | . 2.2 |
| SEND05 | DSBL | | BIF DISABL |
| | L | R2,MCW | . 2.0 |
| | AIM | R2,-2 | . 1.6 |
| | ST | R2,MCW | . 2.2 |
| | DL | RW,CW*RCV,R15 | . 2.4 |
| | DST | RW,CWRcv,R2 | . 2.6 |
| | L | RW,CW*TYP,R15 | . 2.0 |
| | JC | NEZ,NOTIFY | . 1.6/2.0 |
| | L | R2,USTSA | . 2.0 |
| | CIM | R2,IUDTSA | . 29W |
| | JC | LT,SEND1W | . 1.6/2.0 |
| | AIM | R2,2 | . 1.6 |
| | ST | R2,MCW | . 2.2 |
| | L | R1,CWQPTR | . 2.0 |
| | ST | R15,CWQ,R1 | . 2.2 |
| | IM | CWOPTR | . 3.2 |
| | ZR | R15 | . 1.4 |
| | J | MOTIFY | . 2.0 |
| SEND10 | AR | R1,R0 | . 1.4 |
| | ST | R1,CWTRA,R2 | . 2.2 |
| | IM | USTSA | . 3.2 |
| | LR | R1,R0 | . 1.4 |
| | L | RW,MBUF1 | . 2.0 |
| | JC | GZ,SEND15 | . 1.6/2.0 |
| | AIM | R1,MTSAP2-MTSAP1 | . 1.6 |
| SEND15 | L | R0,CW*DADR,R15 | . 2.0 |
| | ST | R0,MTSAP1,R1 | . 2.2 |
| NOTIFY | L | R0,CW*CODE,R15 | . 2.0 |
| | JC | EZ,SEND20 | . 1.6/2.0 |
| | L | R2,MNINDEX | . 2.0 |
| | L | R1,CW*DADR,R15 | . 2.0 |
| | DST | RW,MNCODE,R2 | . 2.0 |
| | AIM | R2,2 | . 1.6 |
| | ANDM | R2,MNMAX | . 1.6 |
| | ST | R2,MNINDEX | . 2.2 |
| SEND20 | ENBL | | BIF ENABLE |
| | L | R15,CW*NXT,R15 | . 2.0 |
| | JC | NEZ,SEND05 | . 1.6/2.0 |
| | L | R2,SENDSV | . 2.0 |
| SEND25 | J | 0,R2 | . 2.0 |
| | EVEN | | |
| SENDSV | STORAGE | 1 | |

32.6 USEC ALLOCATED ENTRY
20.8 USEC ALLOC ADDITIONAL CW
55.2 USEC UNALLOCATED ENTRY
44.8 USEC UNALLOC ADDIT CW
11.6 USEC NOTIFICATION

~~THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC~~

A. 8 QUEUE/DQUEUE - TUNED FOR HOL DEFICIENCIES

| | | | |
|--------|-----------------|---------------------------|---------------|
| QUEUE | EQU | \$ | |
| L | R15,0,R15 | . 2.0 | |
| L | R15,0,R15 | . 2.0 | |
| L | R1,EXCQ | . 2.0 | |
| ST | R1,EXC'NEXC,R15 | . 2.2 | |
| ST | R15,EXCQ | . 2.2 | |
| J | 0,R2 | . 2.0 | 12.4 USEC |
| DQUEUE | EQU | \$ | |
| ST | R2,DQUESV | . 2.2 | |
| DQLOOP | DSBL | | BIF DISABL |
| L | R1,EXCQ | . 2.0 | |
| JC | EZ,DQUE05 | . 1.6/2.0 | |
| L | R0,EXC'NEXC,R1 | . 2.0 | |
| ST | R0,EXCQ | . 2.2 | |
| L | R15,EXC'PRC,R1 | . BIF CEXCPRC | |
| JS | R2,0,R15 | . BIF CEXCPRC | |
| J | DQLOOP | . 2.0 | INLINE DSPTSK |
| DQUE05 | | . | |
| J1 | DQUESV | . | |
| | | 4.0 USEC PLUS | |
| | | 7.8 USEC EACH CEXCPR CALL | |

Appendix B

TUNING THE DAIS MISSION DFG

The formal Dais mission DFG was tuned by systematic application of the techniques discussed in detail in Section 1.2.2. The major stages in the tuning process in order of application were:

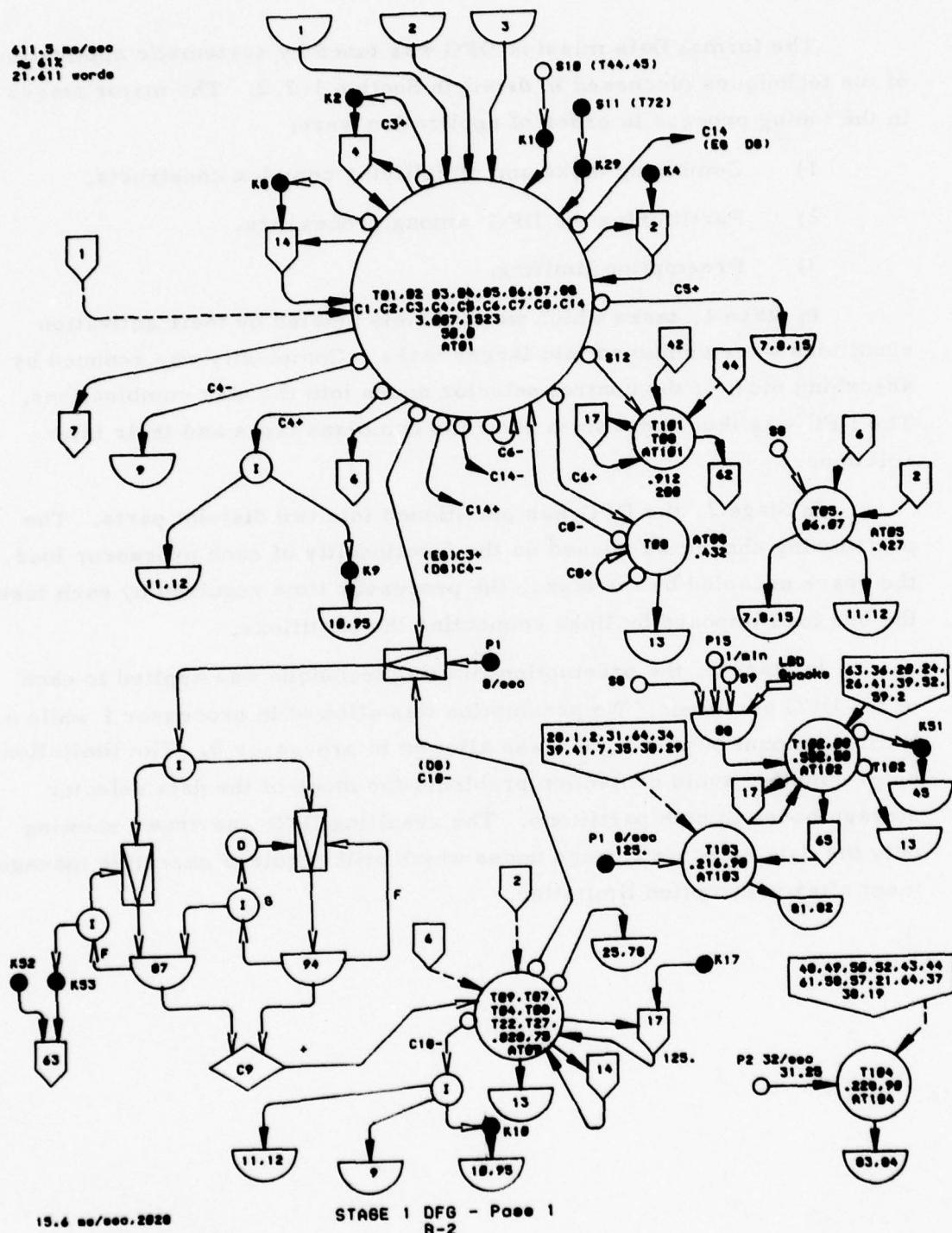
- 1) Combining tasks and simplifying complex constructs.
- 2) Partitioning the DFG among processors.
- 3) Preemption limiting.

In Stage 1, tasks which were closely related by their activation conditions were combined into larger tasks. Complexity was reduced by absorbing most of the control selector nodes into the task combinations. The DFG was then redrawn to show the combined tasks and their inter-relations.

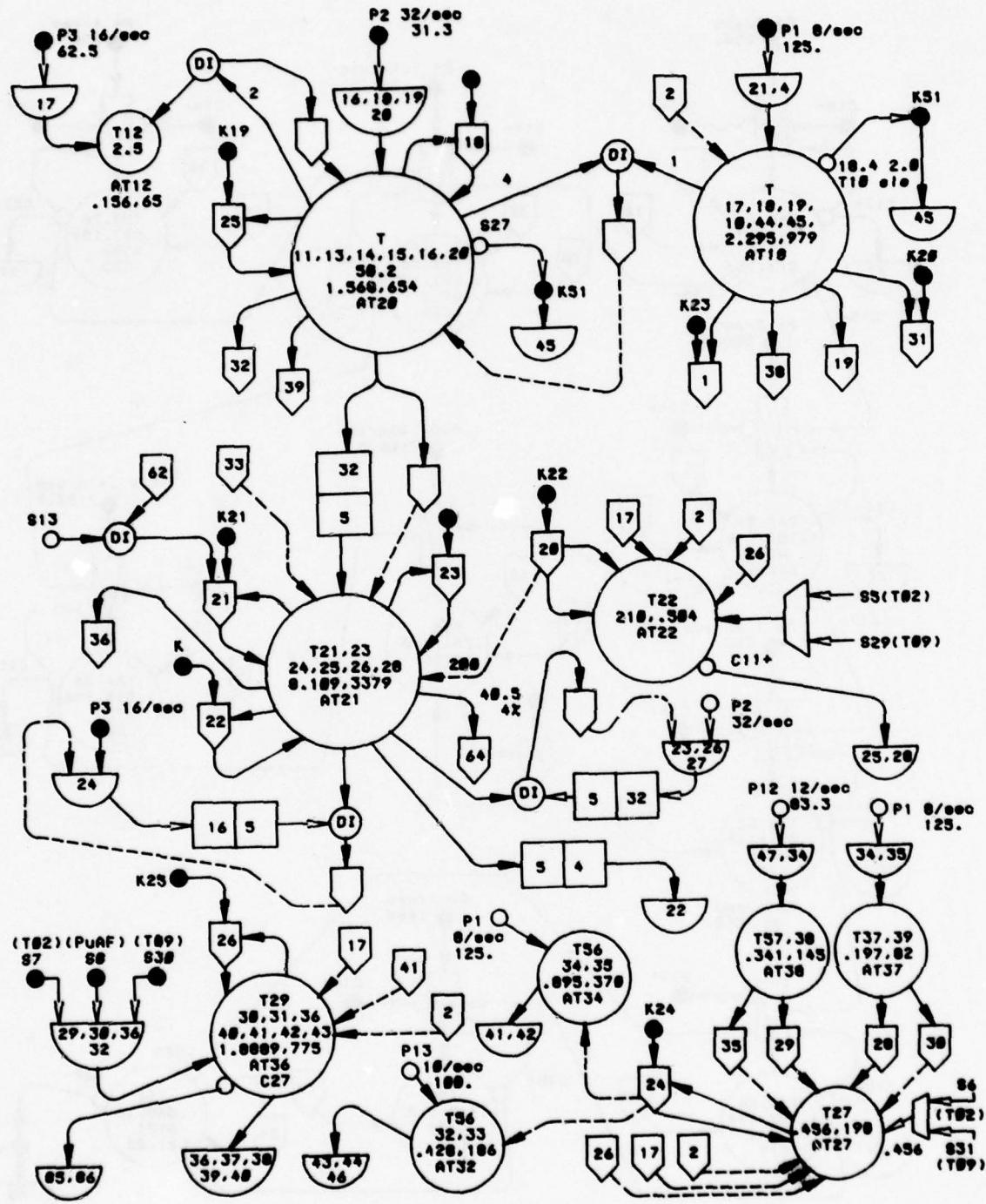
In Stage 2, the DFG was partitioned into two disjoint parts. The partitioning chosen was based on the functionality of each processor load, the space occupied by the tasks, the processor time required by each task, the bus load imposed by links connecting the partitions.

In Stage 3, the preemption limiting technique was applied to each of the DFG partitions. No preemption was allowed in processor 1 while a limited amount of preemption was allowed in processor 0. The limitations on preemption avoid contention problems for most of the data selector storage nodes in both partitions. The resulting DFG was drawn showing only the data selector storage nodes which still required executive management after preemption limitation.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC



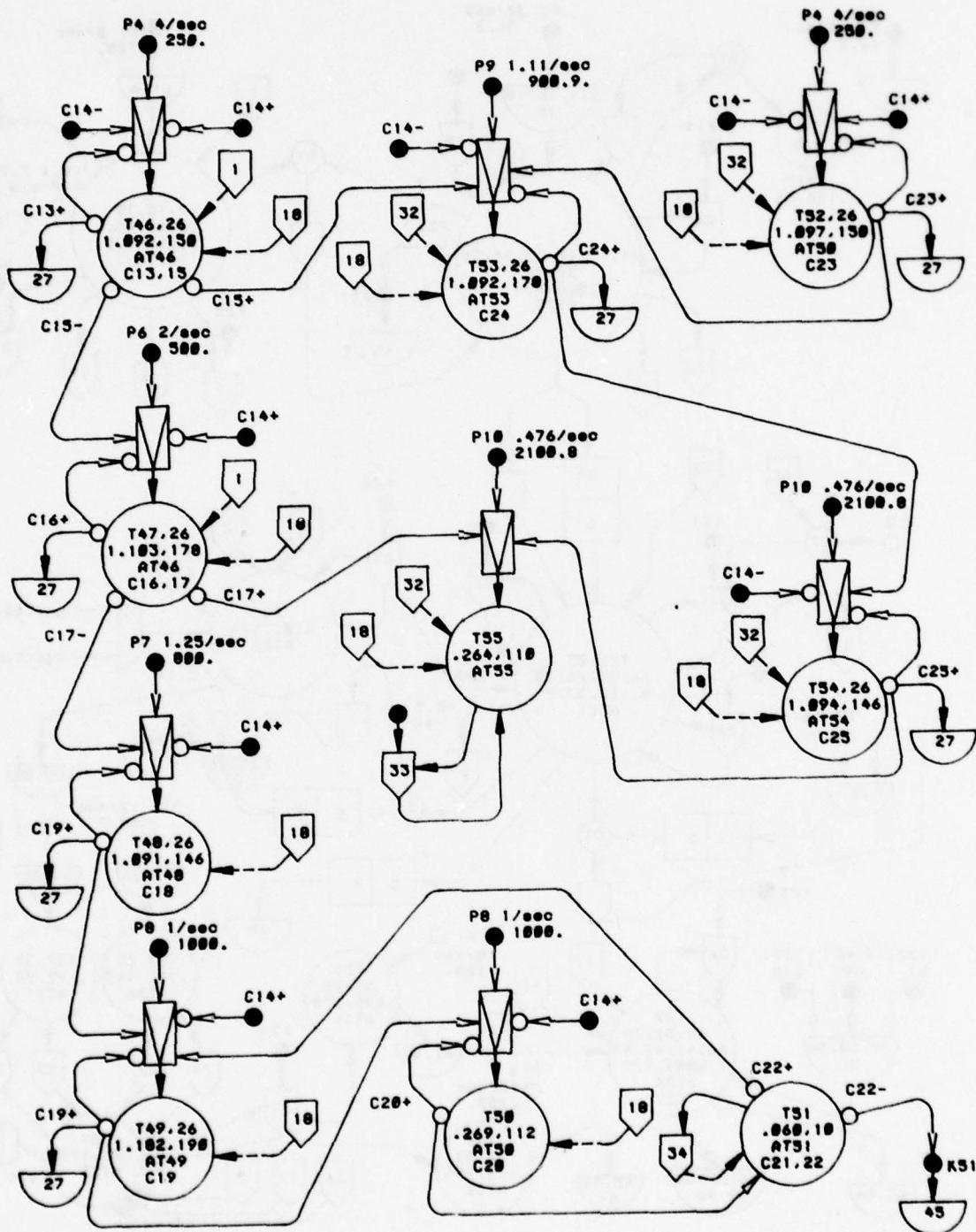
THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC



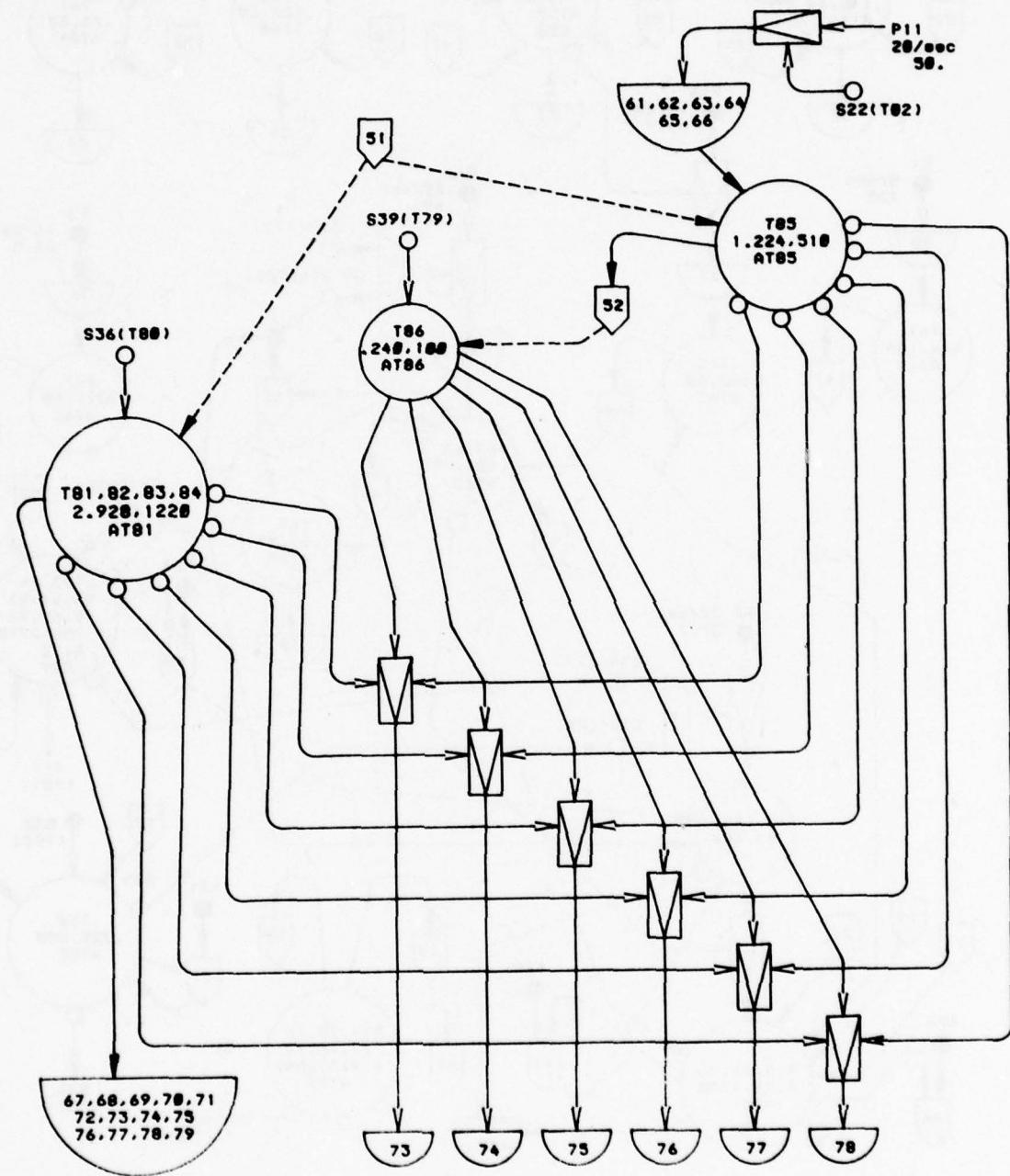
120.7 m/sec., 7035

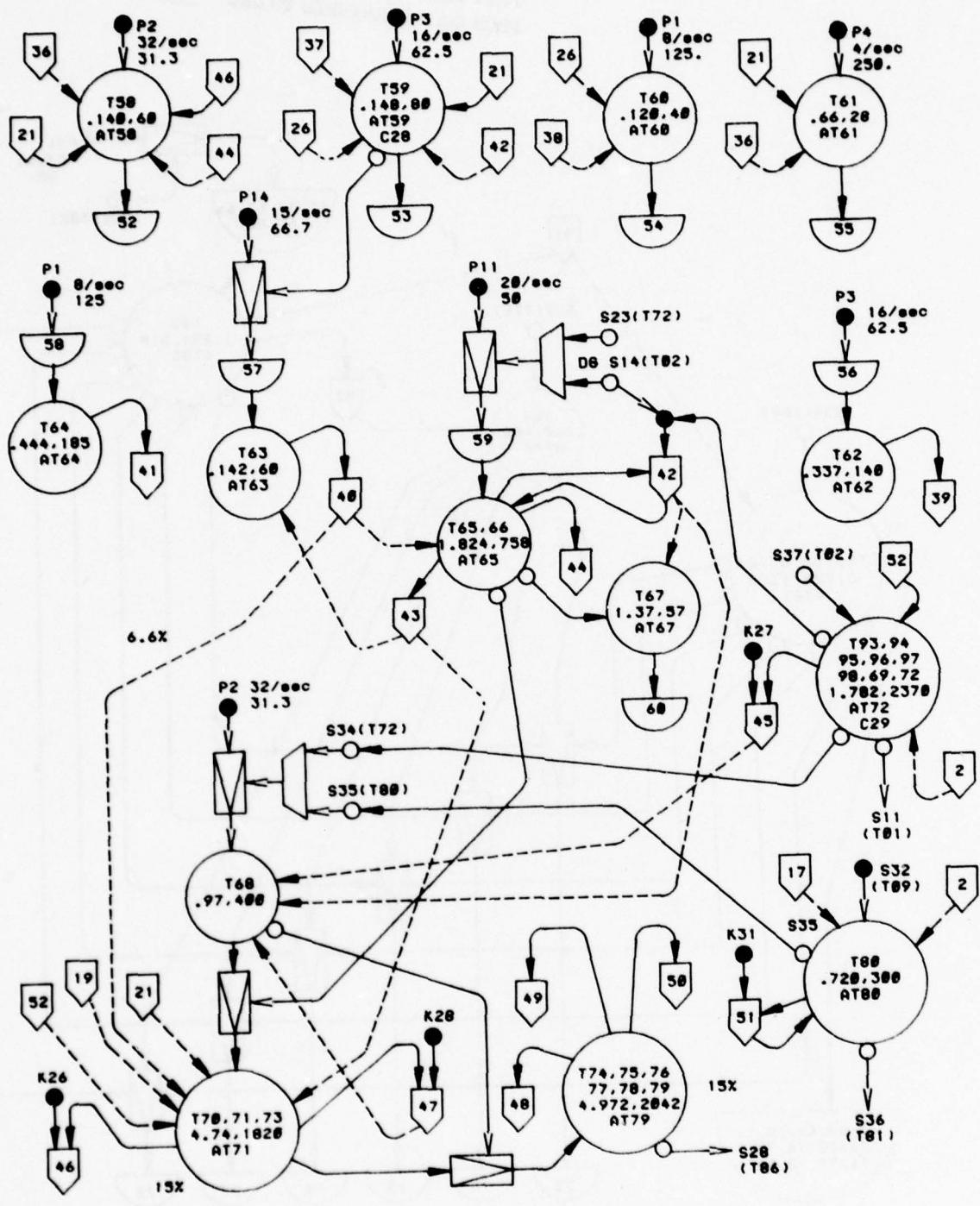
STAGE 1 DF8-Pass 2
B-3

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC



THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

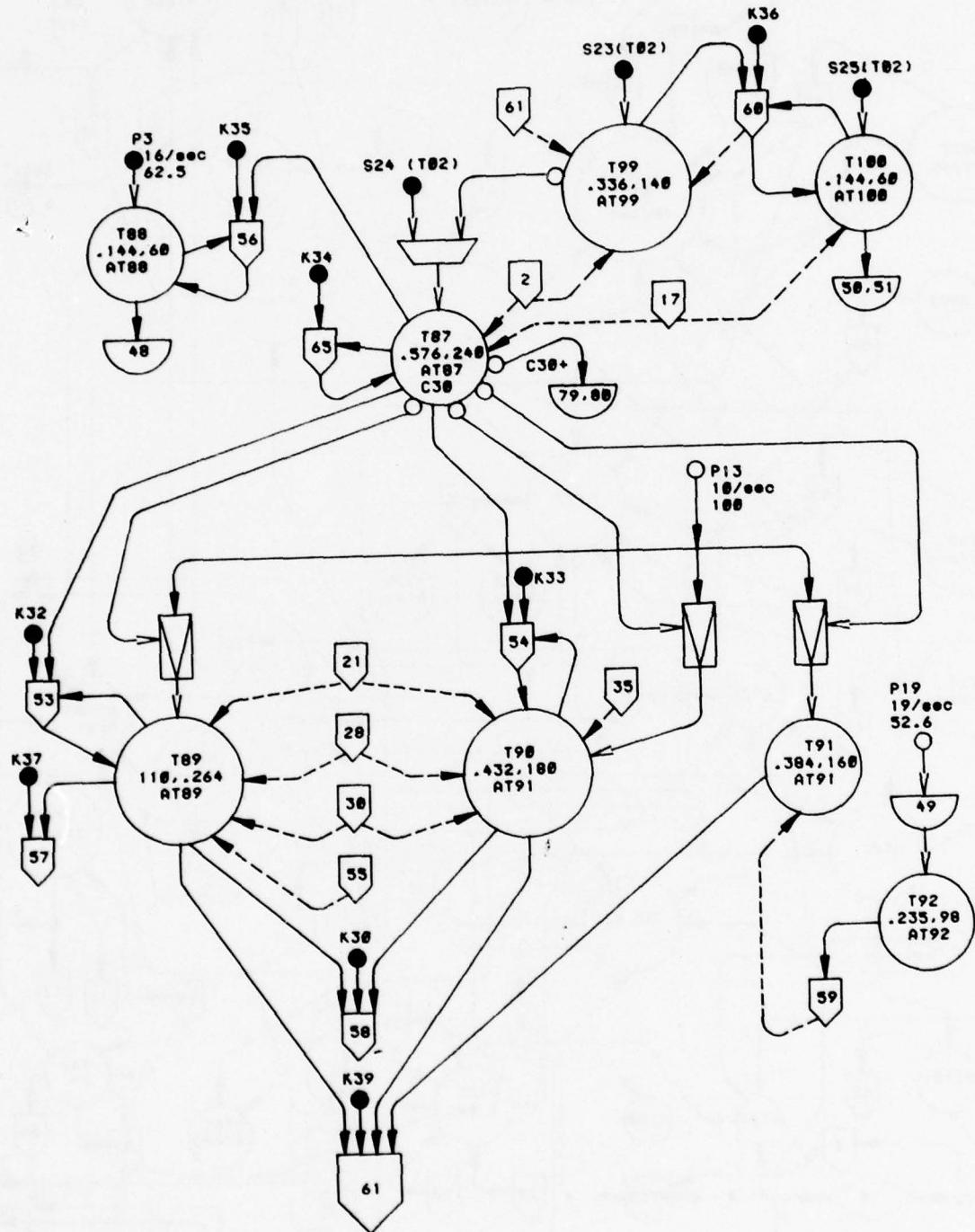




427.2 ms/sec, 8300

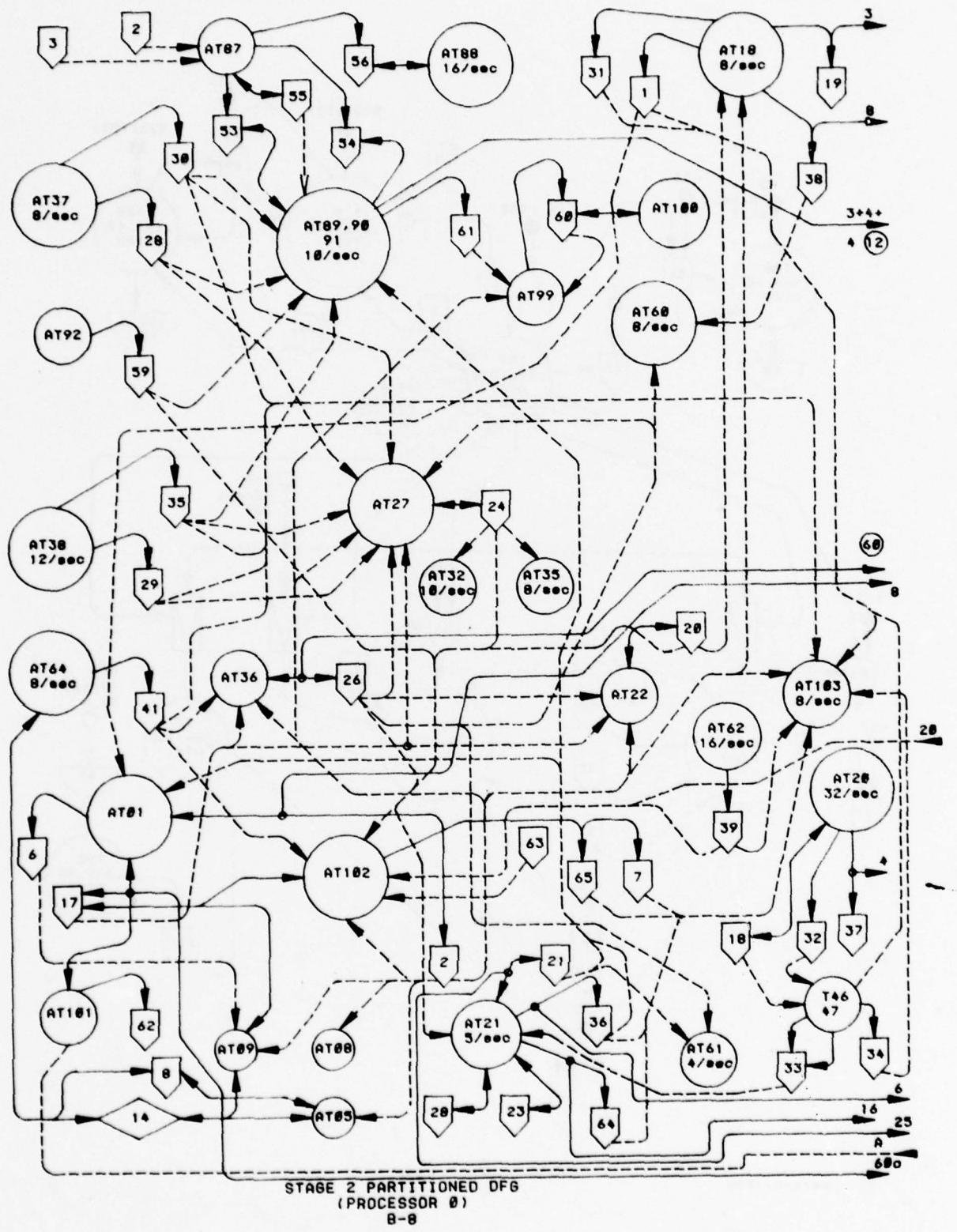
STAGE 1 DFG - Pass 5

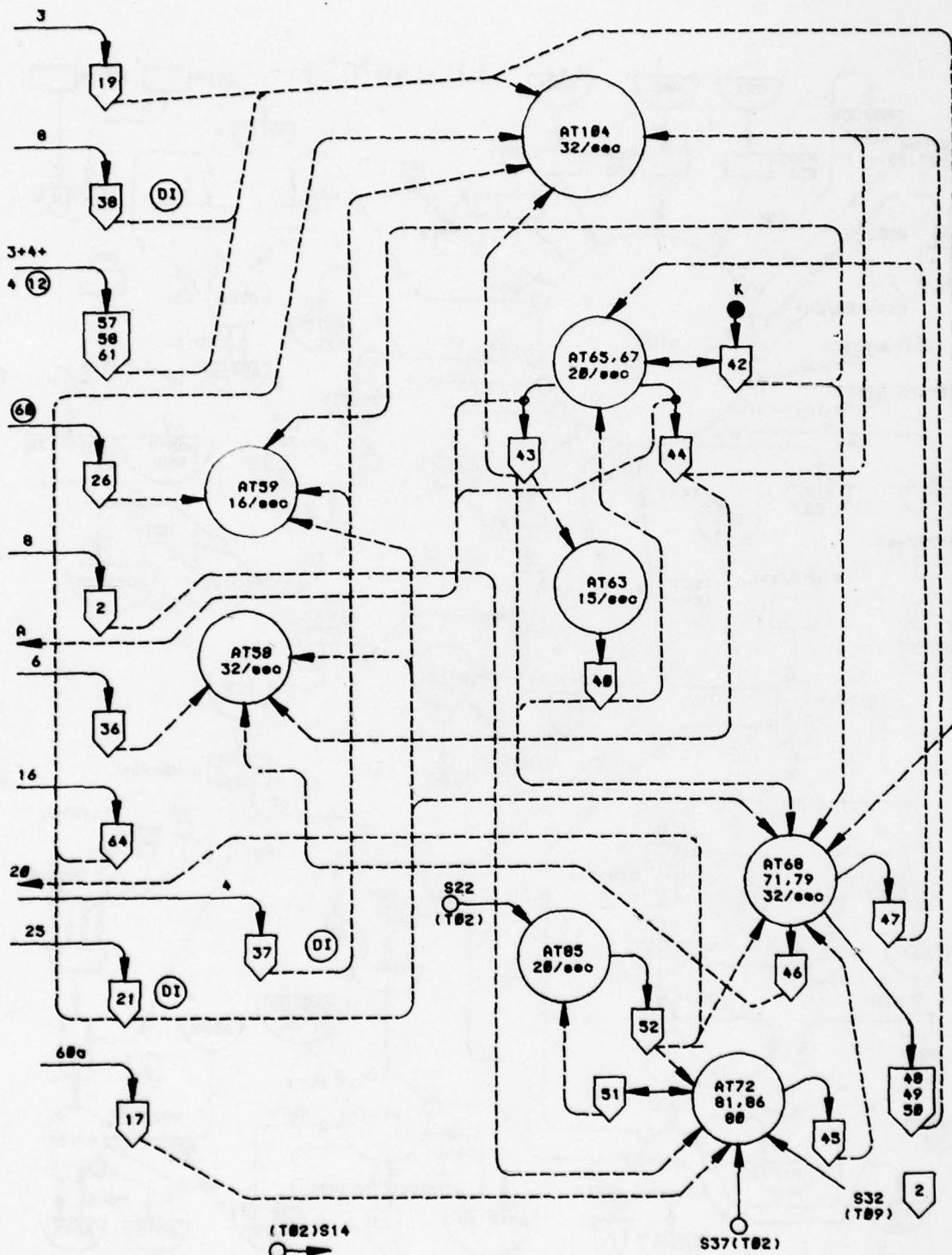
B-6



STAGE 1 DFG - Page 6
B-7

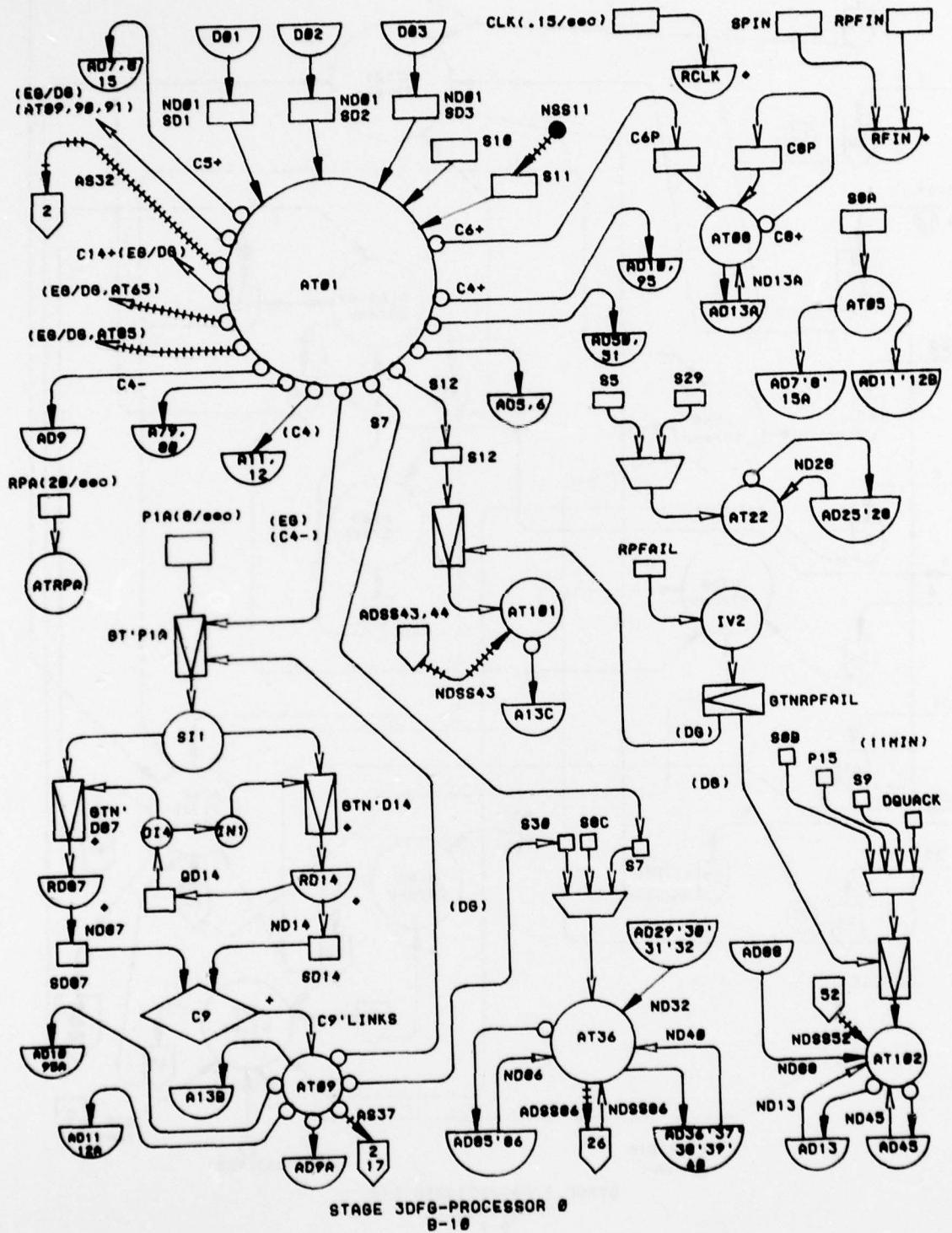
11.1 ms/sec, 1048



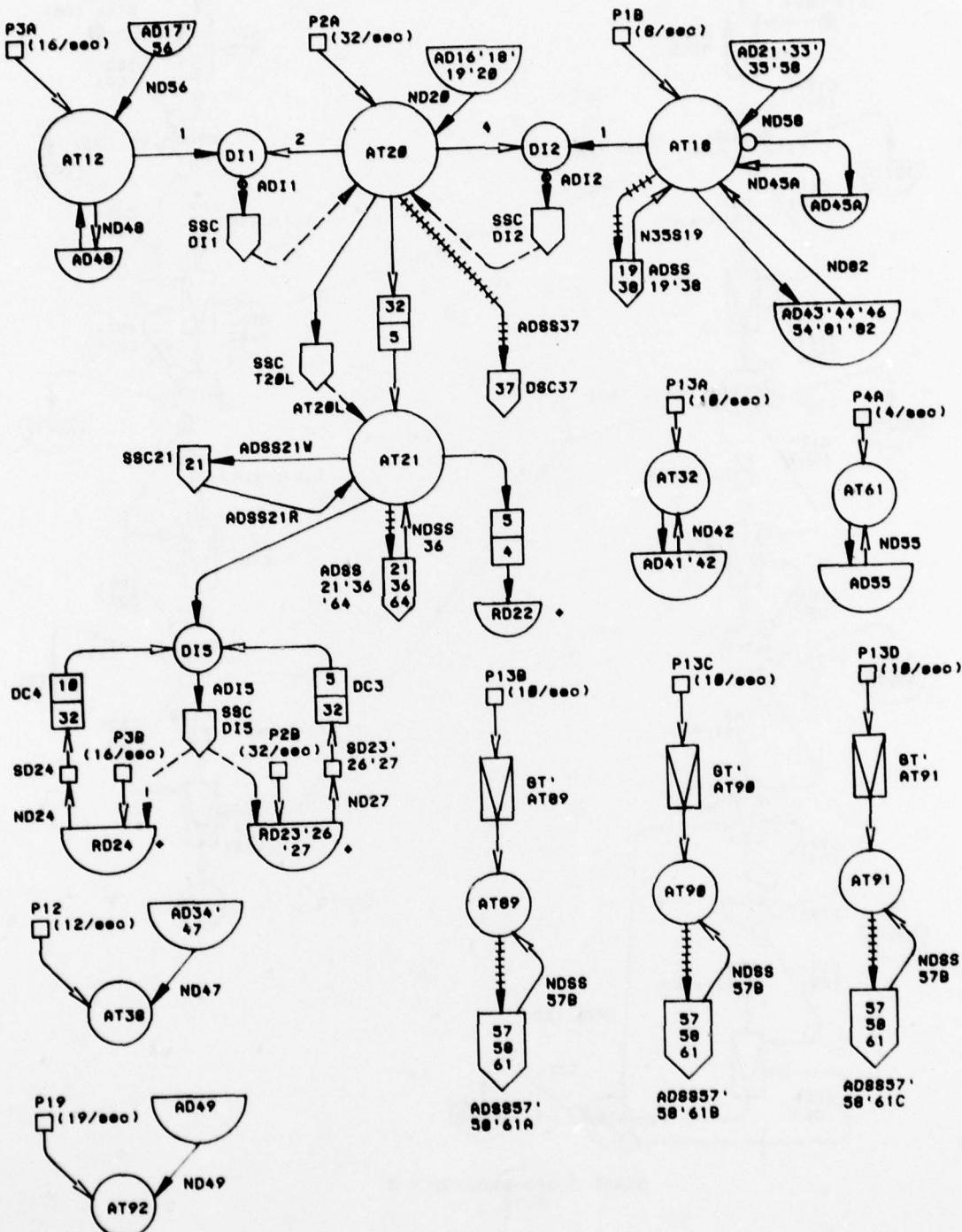


(T82)S14

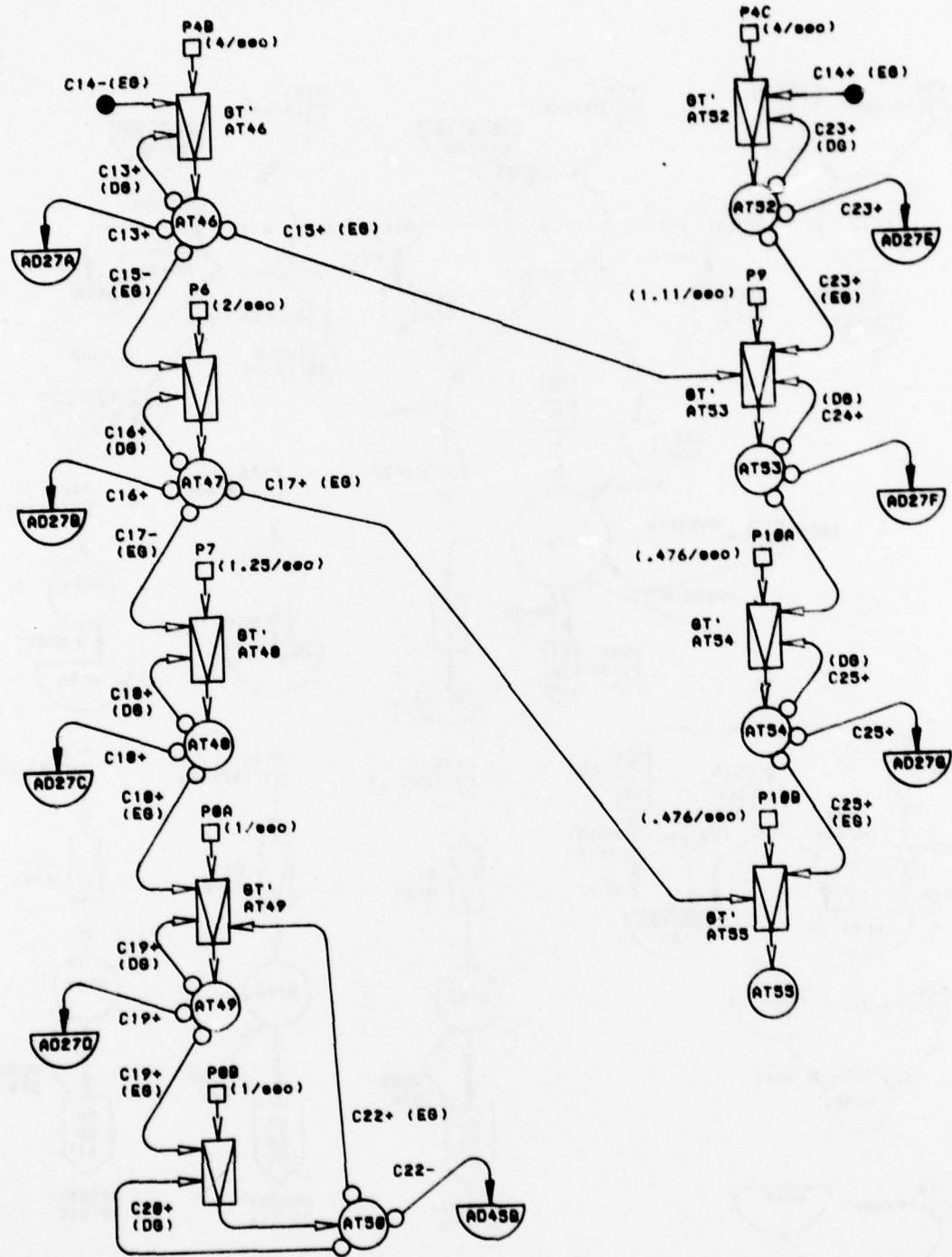
STAGE 2 PARTITIONED DFG
PROCESSOR 1
B-9



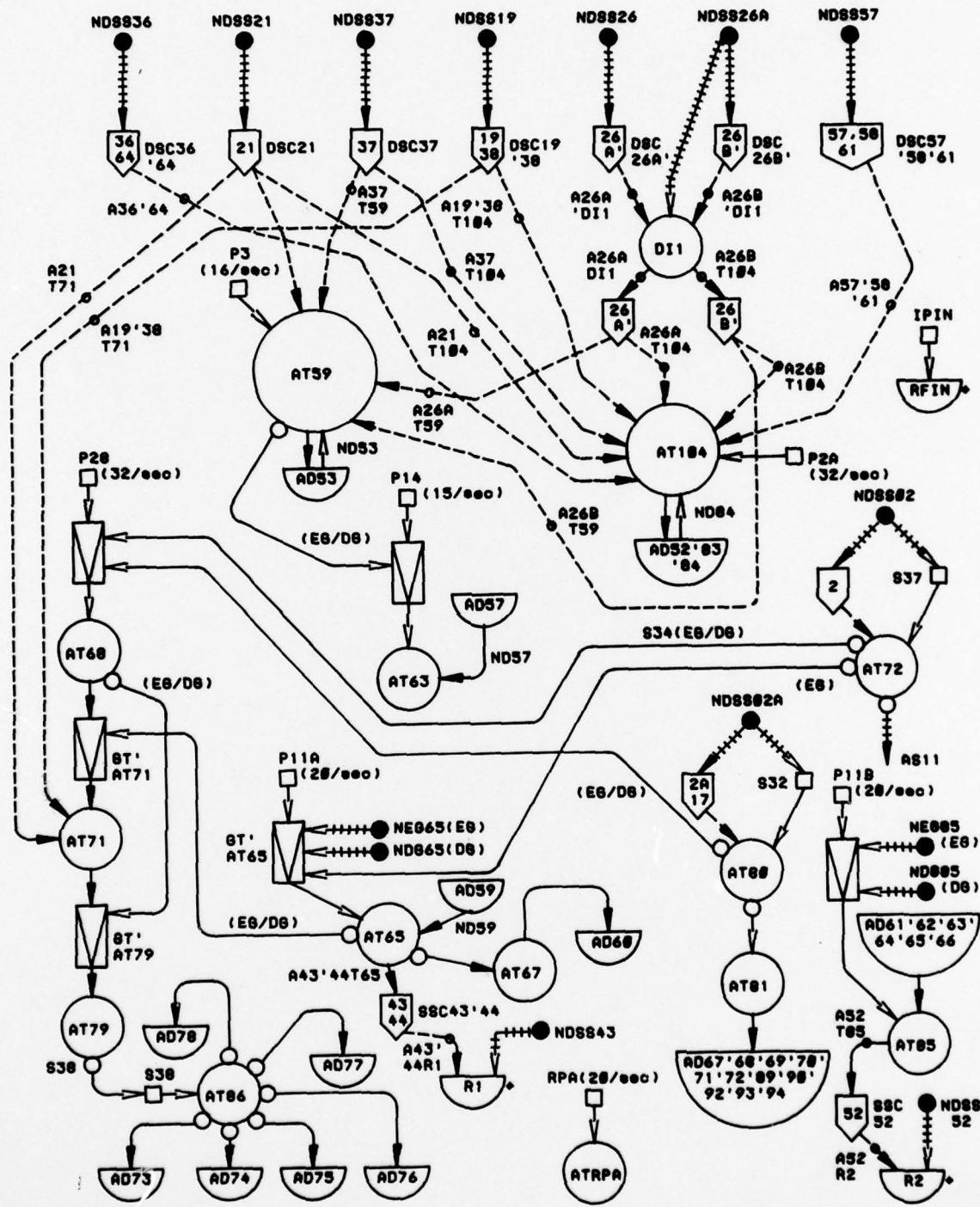
STAGE 3DFG-PROCESSOR 0
B-16



STAGE 3 DFG-PROCESSOR 0
B-11



STAGE 3 DFG-PROCESSOR 8
B-12



**STAGE 3 DFG-PROCESSOR 1
B-13**

Appendix C

ASSIGNMENT OF BUS SUBADDRESSES

The strategy for assignment of subaddresses has been designed to maximize efficiency by using fixed subaddress allocations for the most frequently transmitted messages and dynamic allocation for the numerous less frequently sent messages.

At any given point in time, exactly one processor is in master mode (i. e., in control of the bus) and all others are in remote mode. Each processor has three tables of subaddresses containing the addresses of data areas for the transmittal and receipt of messages. These tables are:

- 1) Master transmit - subaddresses for data to be sent to other processors and remote terminals while in master mode.
- 2) Master receive - subaddresses for data received from remote terminals (excluding other processors) while in master mode.
- 3) Remote receive - subaddresses for data received from other processors while in remote mode.

For a given processor each active sink and outbound link requires a master transmit subaddress, each active source requires a master receive subaddress, and each active inbound data link requires a remote receive subaddress. Subaddresses for any of these cases may be either pre-allocated and fixed or dynamically allocated and variable. Since dynamic allocation takes more time, frequently transmitted or received messages are pre-allocated fixed subaddresses beginning with subaddress 1. The remaining subaddresses belong to a pool which are allocated on a first-come-first-serve basis. Dynamically allocated remote receive subaddresses are allocated by the transmitting master processor.

In any of these cases, buffers for transmission or receipt of data may be dynamically or statically allocated. Dynamic allocation is used for remote receive subaddresses for instances in which the time of receipt is unknown, that is while one copy of received data is in use, another

copy may be received. In these cases a new buffer is allocated for each subaddress where data has been received when bus control is transferred to another processor. All remote receive subaddresses which are dynamically allocated also make use of dynamically allocated buffers.

As mentioned above, the preferred method of subaddress assignment is pre-allocation. In some cases it is not possible to pre-allocate subaddresses for all messages in a given class because of the limited number of subaddresses available (30 not including the subaddress used for transfer of control). A case in point is the master transmit subaddresses for processor 0. Processor 0 contains many sinks and out-bound data links (more than 70). The 16 most frequently transmitted messages have been pre-allocated subaddresses. The remaining messages compete for the remaining subaddresses. It is possible in this case that by coincidence enough transmission requests occur in a short period of time (since the last time the processor was master) to consume all the available master transmit subaddresses. When this happens, subroutine SEND queues the excess requests for execution the next time the processor becomes master.

The following pages detail the subaddress assignments used in each processor.

PROCESSOR O SUBADDRESS ASSIGNMENTS

| | MTSAP | Notify Code |
|-----|------------------------|---------------------------|
| 1. | D23 | |
| 2. | D24 | ND24 |
| 3. | D26 | |
| 4. | D27 | Preallocated ND27 |
| 5. | D41 | Subaddresses |
| 6. | D42 | ND42 |
| 7. | D43 | |
| 8. | D44 | |
| 9. | D46 | |
| 10. | D54 | |
| 11. | D81 | |
| 12. | D82 | ND82 |
| 13. | D48 | ND48 |
| 14. | DSS57A, DSS57B, DSS57C | NDSS57A, NDSS57B, NDSS57C |
| 15. | DSS19(,38) | NDSS19 |
| 16. | DSS57(,58,61) | IUSTSA |
| 17. | | |
| 18. | | |
| 19. | | |
| 20. | | |
| 21. | | Pool of |
| 22. | | Dynamically |
| 23. | | Allocated |
| 24. | | Subaddresses |
| 25. | | |
| 26. | | |
| 27. | | |
| 28. | | |
| 29. | DSS37 | IUDTSA |
| 30. | | |
| 31. | PRIL | |

MASTER TRANSMIT

PROCESSOR 0 SUBADDRESS ASSIGNMENTS

MTSAP

Notify Code

| | |
|------|-------|
| D05 | |
| D06 | |
| D06A | |
| D06B | |
| D07 | |
| D07A | |
| D08 | |
| D08A | |
| D09 | |
| D09A | |
| D10 | |
| D10A | |
| D11 | |
| D11A | |
| D11B | |
| D12 | |
| D12A | |
| D12B | |
| D13 | ND13 |
| D13A | ND13A |
| D13B | |
| D13C | |
| D15 | |
| D15A | |
| D22 | |
| D25 | |
| D27A | |
| D28 | ND28 |
| D36 | |
| D37 | |
| D38 | |
| D39 | |

UNALLOCATED MASTER TRANSMIT

Page 1 of 2

PROCESSOR 0 SUBADDRESS ASSIGNMENTS

| MTSAP | Notify Code |
|----------|-------------|
| D40 | ND40 |
| D45 | ND45 |
| D45A | ND45A |
| D50 | |
| D51 | |
| D55 | ND55 |
| D79 | |
| D80 | |
| D85 | |
| D86 | ND86 |
| D95 | |
| D95A | |
| DSS36,64 | NDSS36 |
| DSS21 | |
| DSS26 | |
| DSS26A | NDSS26A |
| DSS02 | |
| DSS02A | |
| DSS17A | |
| DSS17B | |
| D45B | |
| EG65 | |
| DG65 | |
| EG85 | |
| DG85 | |
| RDSS43 | |
| RDSS52 | |

UNALLOCATED MASTER TRANSMIT

Page 2 of 2

PROCESSOR 0 SUBADDRESS ASSIGNMENTS

| | MRSAP | | Notify Code |
|-----|---------------|----------------|-------------|
| 1. | D01, D02, D03 | | ND01 |
| 2. | D04 | | |
| 3. | D14, D87 | | ND14 |
| 4. | D16 | | |
| 5. | D17 | | |
| 6. | D18 | | |
| 7. | D19 | | |
| 8. | D20 | Statically | ND20 |
| 9. | D21 | Allocated | |
| 10. | D29 | Subaddresses | |
| 11. | D30 | | |
| 12. | D31 | | |
| 13. | D32 | | ND32 |
| 14. | D33 | | |
| 15. | D34 | | |
| 16. | D35 | | |
| 17. | D47 | | ND47 |
| 18. | D49 | | ND49 |
| 19. | D56 | | ND56 |
| 20. | D58 | | ND58 |
| 21. | D88 | | ND88 |
| 22. | | IUDRSA, IUSRSA | |
| 23. | | | |
| 24. | | | |
| 25. | | | |
| 26. | | | |
| 27. | | | |
| 28. | | | |
| 29. | | | |
| 30. | DUMSTDR | | |
| 31. | | | |

MASTER RECEIVE

PROCESSOR 0 SUBADDRESS ASSIGNMENTS

| RRSAP | Notify Code |
|------------------|---------------------------|
| 1. DSS43 | NDSS43 |
| 2. DSS52 | Preallocated NDSS52 |
| 3. SS11 | Subaddresses NDSS11 |
| 4. D01, D02, D03 | Statically Allocated ND01 |
| 5. DRPFIN | Buffers NFIN |
| 6. | RRIMAX |
| 7. | RRADYN, RRUDYN |
| 8. | |
| 9. | |
| 10. | |
| 11. | |
| 12. | |
| 13. | |
| 14. | |
| 15. | |
| 16. | |
| 17. | |
| 18. | |
| 19. | |
| 20. | |
| 21. | |
| 22. | |
| 23. | |
| 24. | |
| 25. | |
| 26. | |
| 27. | |
| 28. | |
| 29. | |
| 30. | |
| 31. PRI1 | |

REMOTE RECEIVE

PROCESSOR 1 SUBADDRESS ASSIGNMENTS

| MTSAP | Notify Code |
|-----------|------------------------------|
| 1. DSS43 | |
| 2. DSS52 | |
| 3. SS11 | Preallocated Subaddresses |
| 4. D52 | |
| 5. D83 | |
| 6. D84 | ND84 |
| 7. D53 | ND53 |
| 8. D60 | |
| 9. D67 | |
| 10. D68 | |
| 11. D69 | |
| 12. D70 | |
| 13. D71 | |
| 14. D72 | |
| 15. D89 | |
| 16. D90 | |
| 17. D91 | |
| 18. D92 | |
| 19. D93 | |
| 20. D94 | |
| 21. D73 | |
| 22. D74 | |
| 23. D75 | |
| 24. D76 | |
| 25. D77 | |
| 26. D78 | |
| 27. DPRIN | IUSTSA, IUDTSA |
| 28. | |
| 29. | |
| 30. | |
| 31. PRIL | |

MASTER TRANSMIT

PROCESSOR 1 SUBADDRESS ASSIGNMENTS

| MRSAP | Notify Code |
|-------------|----------------|
| 1. D57 | ND57 |
| 2. D59 | ND59 |
| 3. D61 | |
| 4. D62 | |
| 5. D63 | |
| 6. D64 | |
| 7. D65 | |
| 8. D66 | ND66 |
| 9. | IUDRSA, IUSRSA |
| 10. | |
| 11. | |
| 12. | |
| 13. | |
| 14. | |
| 15. | |
| 16. | |
| 17. | |
| 18. | |
| 19. | |
| 20. | |
| 21. | |
| 22. | |
| 23. | |
| 24. | |
| 25. | |
| 26. | |
| 27. | |
| 28. | |
| 29. | |
| 30. DUMSTOR | |
| 31. | |

MASTER RECEIVE

PROCESSOR 1 SUBADDRESS ASSIGNMENTS

| | RRSAP | | Notify | Code |
|-----|---------------|--------------|--------|---------|
| 1. | DSS02 | | | NDSS02 |
| 2. | DSS17B | Preallocated | | NDSS02A |
| 3. | EG65 | Subaddresses | | NEG65 |
| 4. | DG65 | Statically | | NDG65 |
| 5. | EG85 | Allocated | | NEG85 |
| 6. | DG85 | Buffers | | NDG85 |
| 7. | RDSS43 | | | NRDSS43 |
| 8. | RDSS52 | | | NRDSS52 |
| 9. | DSS26 | RRADYN | | NDSS26 |
| 10. | DSS26A | | | NDSS26A |
| 11. | DSS37 | Preallocated | | NDSS37 |
| 12. | DSS21 | Subaddresses | | NDSS21 |
| 13. | DSS36(,64) | Dynamically | | NDSS36 |
| 14. | DSS19(,38) | Allocated | | NDSS19 |
| 15. | DSS57(,58,61) | Buffers | | NDSS57 |
| 16. | DSS02A | RRIMAX | | |
| 17. | DSS17A | RRUDYN | | |
| 18. | | | | |
| 19. | | | | |
| 20. | | | | |
| 21. | | Dynamically | | |
| 22. | | Allocated | | |
| 23. | | Subaddresses | | |
| 24. | | Dynamically | | |
| 25. | | Allocated | | |
| 26. | | Buffers | | |
| 27. | | | | |
| 28. | | | | |
| 29. | | | | |
| 30. | | | | |
| 31. | PRIL | | | |

REMOTE RECEIVE